Our Ref. No.: 003394.P007x
Express Mail No. EL651890948US

UTILITY APPLICATION FOR UNITED STATES PATENT

FOR

INSTANCE BROWSER FOR ONTOLOGY

Inventor(s): Marcel Zvi Schreiber
Rannen Yosef Meir

Instance Browser for Ontology

CROSS-RELATED APPLICATIONS

The application is a continuation-in-part of U.S. Patent Application Serial ( ) entitled "Method and System for Collaborative Ontology Modeling" filed on ( ).

FIELD OF THE INVENTION

The present invention relates to a distributed ontology.

BACKGROUND OF THE INVENTION

Ontology is a philosophy of what exists. In computer science ontology is used to model entities of the real world and the relations between them, so as to create common dictionaries for their discussion. Basic concepts of ontology include (i) classes of instances / things, and (ii) relations between the classes, as described hereinbelow. Ontology provides a vocabulary for talking about things that exist.

Instances / Things

There are many kinds of "things" in the world. There are physical things like a car, person, boat, screw and transistor. There are other kinds of things which are not physically connected items or not even physical at all, but may nevertheless be defined. A company, for example, is a largely imaginative thing the only physical manifestation of which is its appearance in a list at a registrar of companies. A company may own and employ. It has a defined beginning and end to its life.

Other things can be more abstract such as the Homo Sapiens species, which is a concept that does not have a beginning and end as such even if its members do.

Ontological models are used to talk about "things." An important vocabulary tool is "relations" between things. An ontology model itself does not include the "things," but introduces class and relation symbols which can then be used as a vocabulary for talking about and classifying things.

Relations / Properties / Attributes

Relations, also referred to as properties, attributes, functions and slots, are specific associations of things with other things. Relations include:

- Relations between things that are part of each other, for example, between a PC and its flat panel screen;
- Relations between things that are related through a process such as the process of creating the things, for example, a book and its author;
- Relations between things and their measures, for example, a thing and its weight.

Some relations also relate things to fundamental concepts such as natural numbers or strings of characters -- for example, the value of a weight in kilograms, or the name of a person.

Relations play a dual role in ontology. On the one hand, individual things are referenced by way of properties, for example, a person by his name, or a book by its title and author. On the other hand, knowledge being shared is often a property of things, too. A thing can be specified by way of some of its properties, in order to query for the values of other of its properties.

Classes

Not all relations are relevant to all things. It is convenient to discuss the domain of a relation as a "class" of things, also referred to as a frame or, for end-user purposes, as a category. Often domains of several relations coincide, for example, the class Book is the domain for both Author and ISBN Number properties.

There is flexibility in the granularity to which classes are defined. Cars is a class. Fiat Cars can also be a class, with a restricted value of a manufacturer property. It may be unnecessary to address this class, however, since Fiat cars may not have special attributes of interest that are not common to other cars. In principle, one can define classes as granular as an individual car unit, although an objective of ontology is to define classes that have important attributes.

Abstract concepts such as measures, as well as media such as a body of water which cannot maintain its identity after coming into contact with other bodies of water, may be modeled as classes with a quantity relation mapping them to real numbers.

In a typical mathematical model, a basic ontology comprises:

- A set **C**, the elements of which are called "class symbols;"
- For each $C \in \mathbf{C}$, a plain language definition of the class C;

- A set **R**, the elements of which are called "relation symbols;"
- For each R∈**R**:
  - a plain language definition of R; and
  - a list of class symbols called the domain of R, written $C_1 \times C_2 \times \cdots \times C_n$;
- A set **F**, the elements of which are called "function symbols;"
- For each f∈**F**:
  - a plain language definition of f;
  - a class symbol called the domain of f; and
  - a class symbol called the co-domain of f; and
- A binary transitive reflexive anti-symmetric relation, **I**, called the inheritance relation on $\mathbf{C} \times \mathbf{C}$.

In the ensuing discussion, the terms "class" and "class symbol" are used interchangeably, for purposes of convenience and clarity. Similarly, the terms "relation" and "relation symbol" are used interchangeably; and the terms "function" and "function symbol" are also used interchangeably. Further, the term attribute is used for a function.

It is apparent to those skilled in the art that if an ontology model is extended to include sets in a class, then a relation on C × D can also be considered as a function from C to sets in D. In this context, a relation may also be considered as a function or an attribute.

If $\mathbf{I}(C_1, C_2)$ then C_1 is referred to as a subclass of C_2, and C_2 is referred to as a superclass of C_1. Also, C_1 is said to inherit from C_2.

A distinguished universal class "Being" is typically postulated to be a superclass of all classes in **C**.

Variations on an ontology model may include:

- Omission of functions, which are in fact a special case of relations;
- Restrictions of relations to binary relations, or of functions to unary functions, these being the most commonly used relations and functions;
- The ability to specify more about properties of relations, such as multiplicity and invertibility.

The notion of a class symbol is conceptual, in that it describes a generic genus for an entire species such as Books, Cars, Companies and People. Specific instances of the species within the genus are referred to as "instances" of the class. Thus "Gone with the Wind" is an instance of a class for books, and "IBM" is an instance of a class for companies. Similarly, the notions of relation symbol and function symbol are conceptual, in that they serve as templates for actual relations and functions that operate on instances of classes.

Class symbols, relation symbols and function symbols are similar to object-oriented classes in computer programming, such as C++ classes. Classes, along with their members and field variables, defined within a header file, serve as templates for specific class instances used by a programmer. A compiler uses header files to allocate memory for, and enables a programmer to use instances of classes. Thus a header file can declare a rectangle class with members left, right, top and bottom. The declarations in the header file do not instantiate actual "rectangle objects," but serve as templates for rectangles instantiated in a program. Similarly, classes of an ontology serve as templates for instances thereof.

There is, however, a distinction between C++ classes and ontology classes. In programming, classes are a template and they are instantiated to create programming objects. In ontology, classes document common structure but the instances exist in the real world and are not created through the class. Also, there is no counterpart to the programming methods, only to the fields, which correspond to functions or attributes.

Ontology provides a vocabulary for speaking about instances, even before the instances themselves are identified. A class Book is used to say that an instance "is a Book." A function Author allows one to create clauses "author of" about an instance. A relation Siblings allows one to create statements "are siblings" about instances. Inheritance is used to say, for example, that "every Book is a PublishedWork". Thus all vocabulary appropriate to PublishedWork can be used for Book.

Once an ontology model is available to provide a vocabulary for talking about instances, the instances themselves can be fit into the vocabulary. For each class symbol, C, all instances which satisfy "is a C" are taken to be the set of instances of C, and this set is denoted B(C). Sets of instances are consistent with inheritance, so that $B(C_1) \subseteq B(C_2)$ whenever C_1 is a subclass of C_2. Relation symbols with domain $C_1 \times C_2 \times \cdots \times C_n$ correspond to relations on $B(C_1) \times B(C_2) \times \cdots \times B(C_n)$. Similarly, function symbols with domain C_1 and co-domain C_2 correspond to functions with domain $B(C_1)$ and co-domain $B(C_2)$. It is noted that if class C_1 inherits from class C, then every instance of C_1 is also an instance of C, and it is therefore known already at the ontology stage that the vocabulary of C is applicable to C_1.

Ontology enables creation of a model of multiple classes and a graph of relationships therebetween. When a class is defined, its attributes are described using handles to related classes. These can in turn be used to look up attributes of the related class, and thus attributes of attributes can be accessed to any depth.

Reference is now made to FIG. 1A, which is an illustration of enterprise knowledge relationships. FIG. 1A illustrates a typical information flow within an enterprise. An enterprise deals with dozens of data formats, including relational and XML formats. The same information such as an invoice must be picked up by different applications and departments.

Reference is now made to FIG. 1B, which is an illustration of graph-like relationships between ontology classes, corresponding to the relationships of FIG. 1A. Shown in FIG. 1B are classes for invoices 110, companies 120, products 130, components 140, measurements 150, credit ratings 160, people 170 and contact information 180. Each class is modeled individually, as well as their inter-relationships. When a specific invoice instance is described formally, users can derive required information from it. For example, a customer can retrieve the invoice electronically into a procurement system. A manufacturing department can view a bill of materials of products sold in order to replenish stock. A finance department can read a Dunn & Bradstreet credit rating of a customer in order to assess credit exposure.

In distinction to ontologies, XML Schema, or the earlier standard document type description (DTD), allow modeling of a single class by defining a syntax for documents that describe elements of that class. XML documents have a tree structure that may include attributes, and attributes of attributes to a pre-determined depth. However, a user cannot look up attributes of attributes to a depth that is not explicitly included in the XML Schema. Thus if the invoice of FIG. 1 is modeled using an XML Schema intended for transfer of the invoice to a customer's procurement system, it is unlikely that manufacturing and finance departments will find the information they require inside an XML document for an invoice. If one were to model a full graph of relations between classes using self-referential XML Schema, such a Schema would be unusable, as no finite instance document would be valid according to the Schema.

It may thus be appreciated that an ontological model provides a more flexible basis for presenting data than does an XML Schema, particularly where the same data must be presented to multiple applications. The rigorous structure imposed by an XML Schema is likely to include only attributes of interest to the particular application for which the Schema was designed.

While XML Schema specifies a finite list of fields (attributes, attributes of attributes), an ontology is a comprehensive model of the world's entities by classes and the relations between them. As such, there are several barriers to the adoption of ontology as a basis for data standards, including:

- No single party can model the whole world, so unlike a centralized XML Schema, a centralized ontology definition is impractical;

- No ontology model will ever be complete or stable, so unlike XML Schema, a notion of backward compatible updates is required;
- XML Schema and the table schema of the world's relational databases have considerable commercial traction, and thus an ontology model that cannot leverage existing XML Schema work and existing relational database schema work is unlikely to have commercial traction – the barrier being that the XML, relational database and ontology models are inherently incompatible.

Prior art ontology systems include the DARPA Agent Markup Language (DAML). Information about DAML is available on the Internet at http://www.daml.org. DAML includes hierarchical relations, and a "meta-level" for formally describing features of an ontology, such as author, name and subject. DAML includes a class of ontologies for describing its own ontologies. DAML includes a formal syntax for relations used to express ranges, domains and cardinality restrictions on domains and co-domains. DAML provides for a "maximum level" class and a "minimum level" class. "Things" is a class for everything, and is a superclass of all classes. "Nothing" is a class for the empty set, and is a subclass of all classes.

Another prior art system is the Resource Description Framework (RDF), developed by the World Wide Web Consortium. Information about RDF is available on the Internet at http://www.w3.org/RDF/. RDF is distributed and contains a grammar for subject-verb-object constructs, but does not contain an ontological model.

Another prior art system is the Knowledge Interchange Format (KIF). Information about KIF is available on the Internet at http://www.logic.stanford.edu/kif/.

Another prior art system is the Ontology Inference Layer (OIL). Information about OIL is available on the Internet at http://www.ontoknowledge.org/oil/. OIL classes are defined "intentionally" rather than "extensionally." This means that OIL classes can be defined in terms of descriptions that specify the properties that objects must satisfy in order to belong to the class. Attributes of classes play a significant role in OIL. Similar to DAML, OIL includes hierarchical relations, and a "second meta-level" for formally describing features of an ontology, such as author, name and subject. Two types of classes are distinguished in OIL: "primitive" (by default) and "defined". For primitive OIL classes, their associated attributes are considered necessary but not sufficient for membership in the class. For example, if the primitive class Elephant is defined to be a sub-class of Animal with a constraint stating that skin-colour must be grey, then all elephants must necessarily be

animals with grey skin, but there may be grey-skinned animals that are not elephants. When a class is "defined" by attributes, its definition is taken be both necessary and sufficient. For example, if the defined class Carnivores is defined to be a sub-class of Animal with the constraint stating that it eats meat, then every carnivore is necessarily a meat eating animal, and every meat eating animal is a carnivore. Intersections and unions of classes are defined in OIL using what are termed class-expressions, so that, for example, Meat AND Fish defines the class of things that are both meat and fish, and NOT Fish can define all things which are not fish.

A general reference on ontology systems is Sowa, John F., Knowledge Representation, Brooks/Cole, Pacific Grove, CA, 2000.

SUMMARY OF THE INVENTION

The present invention provides for a distributed ontology, built up from individual ontology efforts distributed over the web, which in aggregate comprise a global ontology. The present invention enables cross-referencing of definitions, so that a relation defined in one place may have a domain that uses one or more classes defined in one or more other places. Similarly, a class defined in one place may have sub-classes defined in other places.

The present invention provides the ability to build up an ontology in a collaborative way. The physical distribution of different parts of the ontology is arbitrary, and the different parts may reside on the same physical computer or on different physical computers.

A feature of the present invention is the ability to update ontology definitions, by controlling changes made to an ontology so as to ensure backward compatibility. This ensures that a vocabulary that is valid within the framework of a current ontology will continue to be valid with respect to future evolutions of the ontology. Preferably, the present invention uses rules for allowing only a "safe" set of edits to be performed on an ontology.

Thus an ontology may be updated and yet maintain backward compatibility by adding new classes and relations, by adding superclass / subclass inheritance relations, and by extending existing relations and functions. The update feature of the present invention enables enrichment of an ontology without disrupting the available vocabulary. A relation may be extended by enlarging one or more of the classes C_i from its domain $C_1 \times C_2 \times \cdots \times C_n$ to superclasses thereof. A function may be extended by enlarging its domain from a class to a superclass thereof, and/or by reducing its co-domain from a class to a subclass thereof. For example, if the domain of a function Author is extended from a class Books to a larger class Content, the vocabulary "author of the book" remains valid. Similarly, if the co-domain of the function Author is reduced from a class Living Beings to People, the vocabulary "father of the book's author" remains valid.

The present invention further includes the ability to derive XML Schema for particular classes from an ontology, with arbitrary levels of detail (i.e., export to XML Schema). Fundamentally an ontology can be represented as a directed graph whose nodes represent classes and whose edges represent relations. For example, the relation "author of a book" may be an edge going from a class Books to a class People. Such a graph may have closed paths within it, since attributes of classes may refer back to the classes at some level. For example, a book has an author, which is a person, and a person has publications,

which are books. An XML Schema, on the other hand, corresponds to a tree structure. The present invention enables derivation of a tree structure for an XML Schema from a graph structure for an ontology.

Similarly, the present invention includes the ability to glean ontological information and to publish class and relation definitions from an XML Schema and from the table schema of relational databases (i.e., import from XML Schema and relational databases), and from the class hierarchy of an object-oriented application programming interface.

The present invention also includes the ability to create a correspondence, or mapping, between an ontology and an existing XML Schema or relational table schema. Preferably, the mapping identifies certain types in the XML Schema or relational table schema with certain classes in the ontology.

More generally, the present invention describes a format that can be derived from an ontology, referred to as a "View." A View is preferably associated with a specific class in an ontology, and represents a collection of attributes of the class, where the collection of attributes is appropriate for describing or referencing instances of the class in a given situation.

Using the present invention, an ontology serves as a master vocabulary or central dictionary, where other data standards each correspond to a subset. As such, an ontology can be used to guide the creation of transcodings, or mappings, between any of the other standards, by transcoding in and out of the ontology. This is more efficient than mapping each pair as needed, which is error prone and may have to be done n squared times where n is the number of formats.

The present invention includes a novel user interface for representing elements of an ontology, including inter alia classes, relations, functions and instances of classes. Preferably, the user interface of the present invention uses icons to represent classes and instances of classes. In a preferred embodiment, the present invention enables navigating iteratively from an icon representing an instance to an icon representing an attribute of the instance, whereby a window for a class or an instance of a class contains a list of icons for classes corresponding to the attributes thereof. Such a user interface enables one to navigate an ontology through any number of levels.

There is thus provided in accordance with a preferred embodiment of the present invention an instance browser including a repository of class and relation definitions, a server for responding to queries relating to class and relation definitions in the repository, and a graphical user interface including icons for representing instances of classes.

There is additionally provided in accordance with a preferred embodiment of the present invention a method for instance browsing including

managing a repository of class and relation definitions, responding to queries relating to class and relation definitions in the repository, and displaying icons representing instances of classes.

There is moreover provided in accordance with a preferred embodiment of the present invention a distributed ontology system including a central computer comprising a global ontology directory, a plurality of ontology server computers, each including a repository of class and relation definitions, and a server for responding to queries relating to class and relation definitions in said repository, a computer network connecting said central computer with said plurality of ontology server computers, and a graphical user interface including icons for representing instances of classes.

There is further provided in accordance with a preferred embodiment of the present invention a distributed ontology method including managing a plurality of repositories of class and relation definitions, managing a global ontology directory, responding to queries relating to class and relation definitions in at least one repository, and displaying icons representing instances of classes.

The following definitions are employed throughout the specification and claims.

1. Class symbol - a symbol that is a placeholder for a class; e.g., a symbol for a class Books.

2. Class - a set of real world entities whose elements have a common classification; e.g., a class called Books is the set of all books in existence).

3. Relation symbol - a symbol that is a placeholder for a relation; e.g., a symbol for a relation called Author.

4. Relation – as used in mathematics, a subset of a cross product of classes; i.e., a set of tuples (singleton, pair, triplet or, more generally, "tuple") of instances from one or more classes that obey a relationship; e.g., a relation called Author is the list of all pairs (book, person) where the person is an author of the book.

5. Function symbol - a symbol that is a placeholder for a function; e.g., a symbol for a function called Title.

6. Function - an operation that accepts as input an instance of a class and produces as output an instance of the same or another class; e.g., the function Title accepts as input a book and produces as output a text string; namely, the title of the book.

7. Domain - a class whose elements are inputs for a function or relation; e.g., the domain for a function called Title is a class called Books.

8. Co-domain - a class whose elements are used as outputs for a function; e.g., the co-domain for a function called Title is a class called Text Strings.

9. Instance - an element of a class; e.g., Gone with the Wind is an instance of Books.

10. Instance document - an XML term for a document formatted according to an XML Schema. As used in the invention, an XML Schema is associated with a class and its instance documents are associated with instances of the class.

11. Attribute Value – a function value; i.e., an instance that is the output of a function; e.g., an attribute value for Title is the text string "Gone with the Wind".

12. Subclass - a class that is a subset of another class; e.g., a class called Sherlock Holmes Novels is a subclass of a class called Books.

13. Superclass - a class that is a superset of another class; e.g., a class called Books is a superclass of a class called Sherlock Holmes Novels.

14. Inheritance - the binary relationship on the set of all classes, of one class being a subclass of another class.

15. XML Schema - a set of rules governing the syntax of XML documents.

16. Relational database - a linked collection of tables, each having one or more fields, in which fields of a table may themselves point to other tables.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more fully understood and appreciated from the following detailed description, taken in conjunction with the drawings in which:

FIG. 1A is an illustration of enterprise knowledge relationships;

FIG. 1B is an illustration of graph-like relationships between ontology classes, corresponding to the relationships of FIG. 1A;

FIG. 2 is a simplified illustration of a distributed ontology system in accordance with a preferred embodiment of the present invention;

FIG. 3 is a simplified illustration of a mapping from a web of distributed class and relation definitions to a global ontology model, in accordance with a preferred embodiment of the present invention;

FIGS. 4A and 4B are simplified illustrations of export from an ontology to an XML Schema, in accordance with a preferred embodiment of the present invention;

FIG. 4C is a simplified illustration of a View, in accordance with a preferred embodiment of the present invention;

FIG. 5A shows an ontology network with class hierarchies from different domains and functions which relate between them, in accordance with a preferred embodiment of the present invention;

FIG. 5B shows how different domains may be authored by different companies or organizations who are authorities in their field, in accordance with a preferred embodiment of the present invention;

FIG. 5C shows how different authors may have different ownership models for sharing their models, in accordance with a preferred embodiment of the present invention;

FIG. 5D shows how some parts of the ontology may be commercially sensitive and not shared, in accordance with a preferred embodiment of the present invention;

FIG. 5E shows how a user filters domains based on trusted authorship, in accordance with a preferred embodiment of the present invention;

FIG 5F shows how there may be equivalent classes within an ontology, in accordance with a preferred embodiment of the present invention;

FIG. 5G shows how classes and functions are added according to safe transformations only, without disrupting existing uses of a model, in accordance with a preferred embodiment of the present invention;

FIG. 6 shows conceptually that an ontology model acts as a master thesaurus/dictionary to which other data standards may be mapped, in accordance with a preferred embodiment of the present invention;

FIG. 7 shows a relationship between analysis and run-time data management environments, in accordance with a preferred embodiment of the present invention;

FIG. 8 shows how a traditional EAI platform is replaced with a more flexible web approach utilizing an ontology of the present invention;

FIG. 9 is a simplified diagram of gross profit calculation through an ontology model;

FIG. 10 is a simplified block diagram of components of a comprehensive ontology system, in accordance with a preferred embodiment of the present invention;

FIG. 11 is a UML diagram of a preferred version control model;

FIG. 12 is a Unified Modeling Language (UML) diagram of a preferred access control model;

FIG. 13 is a class diagram for an ontology model, in accordance with a preferred embodiment of the present invention;

FIG. 14 is an extension to FIG. 13 accommodating instance structures, in accordance with a preferred embodiment of the present invention;

FIG. 15 is an extension to FIG 13 showing how concepts may be organized according to a hierarchy of packages, in accordance with a preferred embodiment of the present invention;

FIG. 16 is an extension to FIGS. 13 and 15 showing objects that are treated as resources for the purpose of having independent access control, in accordance with a preferred embodiment of the present invention;

FIG. 17 is a UML model for views and their relationship to ontology models;

FIG. 18 is a simplified block diagram of a run-time platform for implementing a preferred embodiment of the present invention;

FIGS. 19A – 19F are illustrations of the use of icons for browsing instances that are organized according to an ontology, and logically defined sets of such instances, in accordance with a preferred embodiment of the present invention;

FIGS. 20A and 20B are illustrations of processes associated with instances of classes, in accordance with a preferred embodiment of the present invention;

FIGS. 21A and 21B are illustrations of a page containing links to various classes, in accordance with a preferred embodiment of the present invention; and

FIGS. 22A – 22E are illustrations of an instance browser for navigating through class and relation definitions, in accordance with a preferred embodiment of the present invention;

FIG. 23 is a simplified block diagram of an architecture for an instance browser, such as the instance browser illustrated in FIGS. 22A – 22E, in accordance with a preferred embodiment of the present invention;

FIGS. 24A – 24D are illustrations of an ontology authoring tool for creating an ontology model, in accordance with a preferred embodiment of the present invention;

FIGS. 25A – 25E are illustrations of creating classes, properties, functions and inheritance within an ontology model, in accordance with a preferred embodiment of the present invention;

FIGS. 26A – 26F are illustrations of a class viewer with tabs for viewing class information, in accordance with a preferred embodiment of the present invention;

FIG. 27 is an illustration of displaying a relation within an explorer-type viewing tool, in accordance with a preferred embodiment of the present invention;

FIG. 28 is an illustration of a search tool for finding a desired ontology, in accordance with a preferred embodiment of the present invention;

FIG. 29 is an illustration of a tool for viewing ontologies, in accordance with a preferred embodiment of the present invention;

FIG. 30A and 30B are illustrations of a tool for displaying and editing enumerated values of a class, in accordance with a preferred embodiment of the present invention;

FIG. 31A is a diagram of the relationship between a view of a class and a description of an instance and, correspondingly, an XML Schema and an XML document, in accordance with a preferred embodiment of the present invention; and

FIG. 31B is a UML diagram corresponding to FIG. 31A, in accordance with a preferred embodiment of the present invention.

LIST OF APPENDICES

Appendices A1 – A3 are simplified listing of XML documents for example ontologies, in accordance with a preferred embodiment of the present invention;

Appendix B is a simplified listing of an XML Schema for an ontology, in accordance with a preferred embodiment of the present invention; and

Appendices C1 – C6 are simplified listings of example views, corresponding XML schema and corresponding XML documents, in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

The present invention relates to a distributed approach to ontology. Recognizing that one player cannot model an entire world, the present invention enables an ontology to be built up from distributed patchworks of ontology efforts with different owners, which nonetheless in aggregate represent a single global ontology.

Ontology Network

Managing a distributed ontology requires solutions to problems that do not arise with single owner ontologies. As described in detail hereinbelow, such problems and the solutions provided by the present invention include:

Problem: Redundancy

Description: More than one player may provide alternative models for the same part of the real world.

Solution: The distributed ontology of the present invention allows for such alternatives to co-exist, and provides tools for parties to negotiate the ontology to use and for providing mappings between alternative models.

Problem: Reliability

Description: As a global ontology is an open ontology, it may comprise a web of ontological elements originating from owners with different credentials.

Solution: The distributed ontology of the present invention allows for evaluation and filtering of ontologies based on credentials of their owners.

Problem: Evolvability

Description: The physical world, its classes of entities and the relationships between them are constantly changing.

Solution: The distributed ontology of the present invention includes tools for extending an ontology while maintaining backward compatibility.

Problem: Compliance with Standards

Description: Existing documents and data, such as XML Schema and relational database schema, already contain ontological information, although they do not obey a formal ontological model.

Solution: The distributed ontology of the present invention builds upon such existing documents and data. It provides for importing XML Schema and

relational database schema into an ontology model, and for exporting XML Schema and relational database schema from an ontology model.

Problem: Tools

Description: In order to make a distributed ontology effective, there must be a platform of tools making it easy to build and use a global ontology via the web.

Solution: The present invention provides a platform of such tools.

Reference is now made to FIG. 2, which is a simplified illustration of a distributed ontology system in accordance with a preferred embodiment of the present invention. The system shown in FIG. 2 enables multiple parties to publish separate definitions of classes and relations on the web, and update them from time to time. Moreover the relations can use classes from different owners as domains, thereby creating a web.

Shown in FIG. 2 are three ontology server computers, 210, 220 and 230. Each ontology server computer maintains its own ontology in a repository of class and relation definitions, labeled 213, 223 and 233, respectively. Ontology server computers 210, 220 and 230 are shown interconnected by dotted lines, indicating that the server computers may not communicate with each other, but the class and relation definitions in any one of the repositories can contain references to direct a client to the other repositories.

An ontology server may be realized by way of a standard web server, serving the ontology repository using a syntax similar to that documented using XML Schema in Appendix B hereinbelow.

Each repository is made accessible by a server, labeled 216, 226 and 236, respectively. Preferably, each ontology server computer further includes a publisher, labeled 219, 229 and 239, respectively, through which it can publish class and relation definitions in its repository to a global ontology directory 243 residing in a central computer 240, and thereby share them with others. Preferably, ontology directory 243 maintains a master catalogue with links to classes and relations in the various ontologies present in repositories 213, 223 and 233. The master catalogue in global ontology directory 243 may be distributed hierarchically among multiple levels of sub-directories 244. Sub-directories 244 may reside within central computer 243, or alternatively, they may reside in separate computers.

Global ontology directory 243 is managed by a directory manager 246. As shown in FIG. 2, ontology server computer 230 preferably publishes its ontology to global ontology directory 243 through directory manager

246. Upon doing so, global ontology directory 243 modifies its catalogue so as to acknowledge the classes and relations residing on ontology server 230.

Ontology server computers 210, 220 and 230 may be physically distinct computers, or may be the same physical computer. In fact, ontology server computers 210, 220 and 230 may coincide with central computer 240.

Global ontology directory 243 may be realized by:

- A standard web search engine that indexes classes by keyword;
- A Java / C++ program that recreates an ontology web centrally using objects; or
- A relational database with tables for classes, relations, functions and inheritance that list (i) domains for relations, (ii) domains and co-domains for functions, and (iii) subclasses and superclasses for inheritance. Preferably relations are indexed by name, domain and subdomain; and inheritance is indexed by subclasses and superclasses.

Also shown in FIG. 2 is a toolkit 255 for enabling a client computer 250 to use the distributed ontology of the present invention. Toolkit 255 conducts searches of global ontology directory 243 for specific ontological information. After locating an ontology server computer, such as ontology server computer 220, containing the requested information, client computer 250 uses toolkit 255 to query repository 223 and obtain detailed information about classes of interest.

As may be appreciated from FIG. 2, the present invention extends an abstract ontology model to a model for a distributed ontology. Reference is now made to FIG. 3, which is a simplified illustration of a mapping from a web of distributed class and relation definitions to a global ontology model. The distributed ontology allows an ontology model to be built up from an ontology web 310 of distributed sets of class and relation definitions. Taken as a whole, the distributed sets of class and relation definitions define a global ontology model 320. The mapping illustrated in FIG. 3 is referred to as an ontology web semantic mapping, as it helps give meaning to the ontology web.

The data repositories of an ontology web may be linked to one another. Thus a class defined within a specific repository may be a subclass of a referenced class defined within a different repository. Similarly, a function defined within a specific repository may have as its domain a referenced class defined within a different repository, and as its co-domain a referenced class defined within yet another repository. According to the present invention, such referencing can occur whether the repositories reside on one computer or whether they are physically distributed.

The present invention preferably uses a "class definition" as a common way of describing a class. A class definition is a document or segment of a document with the following elements:

Compulsory:

- A natural language description delineating what comprise the elements of the class.

Optional:

- A natural language name / label.
- A set of class references representing superclasses.
  - For each superclass, a list of restrictions on relations with that superclass as domain.
- A set of class references representing subclasses.
- Restrictions (e.g., classes which this class is disjoint from).
- A URL of an icon representing the class.

A feature of the present invention is the ability to update ontology definitions, by controlling changes made to an ontology so as to ensure backward compatibility. This ensures that a vocabulary that is valid within the framework of a current ontology will continue to be valid with respect to future evolutions of the ontology. Preferably, the present invention uses rules for allowing only a "safe" set of edits to be performed on an ontology.

Thus an ontology may be updated and yet maintain backward compatibility by adding new classes and relations, by adding superclass / subclass inheritance relations, and by extending existing relations and functions. The update feature of the present invention enables enrichment of an ontology without disrupting the available vocabulary.

A class definition A is an "update" of a class definition B if the set of superclasses and the set of subclasses in A respectively contain the set of superclasses and the set of subclasses in B, and the description and label are unchanged. As mentioned hereinabove, an update mechanism enables an ontology to dynamically evolve and expand while maintaining backward compatibility.

Specifically, class definitions may be updated so as to include new superclass / subclass relations, and new class and relation definitions may be added to an ontology, without breaking backward compatibility.

A "class server" is a device that can transmit a class definition on demand, and which is preferably committed to not change the class definition it transmits other than by way of update. The class server allows a definition to be available over time and to be updated while still referring to the same class of real

world items. A class server preferably indicates class server metadata, including inter alia:

- Promise of reliability (e.g., in terms of % uptime);
- Promise of consistency, including:
  - No change; and
  - Updates allowed;
- Author;
- Version number;
- Date of last change;
- Accessibility and pricing information; and
- Digital signature on the above information, for authentication purposes, to ensure that an author committing to this behavior is identifiable and remains the same when the class definition is updated.

A "class reference" is a pointer to a class definition. Preferably, there are at least two types of class references: (i) a local class reference, which points to a class definition within the same document in which the reference appears, and (ii) a global class reference, which contains a URL of a class server and a reference to a class within a document at the URL.

Similar to a class definition, the present invention preferably uses a "relation definition" as a common way of describing a relation. A relation definition is a document or segment of a document with the following elements:

Compulsory:

- A natural language description describing the meaning of the relation.
- A list of class references called the domain.

Optional:

- A natural language name / label.
- A list of restrictions on the relation (e.g., whether it is a function, the cardinality of the relation)
- A list of restrictions on the relation when limited to some subclass of the domain.

"Function definitions" are defined similar to relation definitions, when functions are distinguished from relations as in the ontology model described hereinabove.

A relation definition Q is an "update" of a relation definition R if the classes comprising the domain of Q are superclasses of the respective classes comprising the domain of R and, within the domain of Q, Q contains R. As pointed out above, an update mechanism enables an ontology to dynamically evolve and expand while maintaining backward compatibility.

Similarly, a function f is an "update" of a function g if (i) the domain of f is a superclass of the domain of g; and (ii) the co-domain of f is a subclass of the co-domain of g.

The concepts of a "relation server" and a "relation reference" are defined analogously to those of a class server and a class reference.

Given a web of class and relation servers, these at any one time represent a global ontology model. The ontology web semantic mapping defined hereinabove with reference to FIG. 3 is built up as follows:

- The set of classes is the set of valid class definitions available from class servers at any one time;
- The set of relations is the set of valid relation definitions available from relation servers at any one time; and
- The inheritance relation includes all pairs of classes (C_1, C_2) such that either (i) C_1 lists C_2 as a subclass; or (ii) C_2 lists C_1 as a superclass.

Preferably, the inheritance relation of c_1 and c_2 is not listed in either class definition, but rather in a separate inheritance document that contains references to a subclass and to a superclass. In this way, a third party can publish an inheritance relation between two classes on the web. Such an inheritance document enables other third parties to note inheritance between two different classes.

It may be appreciated by those skilled in the art that a function can be considered as a special type of binary relation, and, vice versa, a relation can be considered as a set-valued function. Specifically, if f is a function with domain C and co-domain D, then f can be considered as a relation R on C×D; namely, the subset $\{(a,b)\in C\times D: f(a) = b\}$. If R is a relation on C×D, then R can be considered as a function with domain C and co-domain the class of subsets of D, referred to as the "power set" of D; namely, the function defined by $f(a) = \{b\in D: R(a,b)\}$. Thus, for example, a syntax that provides a description for functions can also be used to described relations, and, vice versa, a syntax that provides a description for relations can also be used to describe functions.

In a preferred embodiment of the present invention, classes, relations, functions and inheritances are described in a syntax for XML documents that conform to a general XML Schema for ontologies. An example of such an ontology XML Schema is provided hereinbelow in Appendix B. Instances are also preferably described using a syntax for XML documents, as described hereinbelow.

Preferably, the present invention allows documents that include ontological information to be implicitly embedded within an ontology web, including, for example, XML Schema, relational database schema, RDF schema

and even simple text documents describing a class. Standards that are embodied in such documents can be included within the ontology web of the present invention. In a preferred embodiment of the present invention, such documents can be posted directly within an ontology repository, and the class and relation definitions implicit therein are identified. In an alternate embodiment of the present invention, such documents are converted to a specific format suited for ontologies.

Preferably, the present invention also allows ontological information to be exported to XML Schema, relational databases, RDF schema and simple text documents. Reference is now made to FIGS. 4A and 4B, which are simplified illustrations of export from an ontology to an XML Schema, in accordance with a preferred embodiment of the present invention. Fundamentally, the classes and relations within an ontology can be depicted as a directed graph, with nodes corresponding to classes and edges corresponding to relations. Such a graph can have closed paths, since properties of a class can lead back to the same class.

Shown in FIG. 4A is a directed graph 400, containing a class 405 for Libraries. Libraries are indicated with two properties; namely, an Address and Contents. The edge for Address points from Libraries to a class 410 for Locations, meaning that the address of a library is a location. The edge for Contents points from Libraries to a class 415 for Books, meaning that the contents of a library are books. Similarly, Books are indicated as having two properties; namely, an Author and an ISBN Number. The edge for Author points from Books to a class 420 for People, meaning that the author of a book is one or more people. The edge for ISBN Number points from Books to a class 425 for Numbers, meaning that the ISBN number of a book is a number. Class 425 for Numbers is a fundamental class; i.e., it corresponds to a fundamental concept.

Similarly, People are indicated as having three properties; namely, Name, Address and Children. The edge for Name points from People to a class 430 for Text Strings, meaning that the name of a person is a text string. Class 430 for Text Strings is also a fundamental class. The edge for Address points from People to class 410 for Locations, meaning that the address of a person is a location. The edge for Children points from People back to People, meaning that the children of a person are zero or more people. Similarly, Locations are indicated as having a property of Contents. The edge for Contents points from Locations to Libraries, meaning that the contents of a location include zero or more Libraries.

Fundamentally, the tags and attributes within an XML Schema can be depicted as a tree. In order to extract an XML Schema from an ontology,

it is therefore necessary to map the directed graph structure of an ontology into the tree structure of an XML Schema.

Shown in FIG. 4B is a tree 450 that is extracted from graph 400 (FIG. 4A), for an XML Schema for Libraries. Tree 450 has a root 455 for Libraries, corresponding to node 405 in graph 400. Root note 455 has two child nodes; namely, node 460 for Locations, corresponding to node 410 in graph 400, and node 465 for Books, corresponding to node 415 in graph 400. In turn node 465 has two child nodes; namely, node 470 for People, corresponding to node 420 in graph 400, and node 475 for Numbers, corresponding to node 425 in graph 400. Finally, node 470 has two child nodes; namely, node 480 for Text Strings, corresponding to node 430 in graph 400, and node 485 for Locations, corresponding to node 410 in graph 400.

It is noted that tree 450 could be continued indefinitely, since Locations can contain Libraries as a child. In order to determine how far tree 450 should extend, preferably a user assists the process of exporting tree 450 from graph 400. The user can choose which relations are to be included in tree 450 and which are not to be included. In tree 450, for example, the user has decided not to include Libraries as a child of Locations in nodes 460 and 485.

In a preferred embodiment of the present invention, an Explorer-type navigational tool is used by a user to select the relations that are to be included in tree 450. A list of classes is displayed, each of which can be expanded an additional level to view the relations thereof. The user selects those relations he chooses to incorporate in the XML Schema.

It is apparent from FIG. 4B, that trees for XML Schema can be derived from one another, so that if an XML Schema for People has already been defined, it may be embedded as a sub-tree in an XML Schema for Libraries by using it instead of node 470.

Tree 450 can be directly converted into an XML document with structure:

```
<Library>
    <Address>
        Location of library
    </Address>
    <Contents>
        Name of book
        <Author>
            <Name>
                Name of author(s)
```

```
                    </Name>
                    <Address>
                        Address(es) of author(s)
                    </Address>
                </Author>
                <ISBN Number>
                    ISBN number of book
                </ISBN Number>
            </Contents>
</Library>
```

Although one can derive an XML Schema corresponding to tree 450, for purposes of clarification a sample XML instance document according to such a Schema has been described above, rather than the formal XML Schema itself. It may be appreciated by those skilled in the art that XML Schema includes DVD and other conventional schematic formalisms.

An XML Schema is but one particular syntax for a more abstract format that can be derived from an ontology, referred to as a "View." A View is preferably associated with a specific class in an ontology, and represents a collection of attributes of the class, where the collection of attributes is appropriate for describing or referencing instances of the class in a given situation. It may be appreciated that one class can be associated with several Views, which can be used to describe the class instances in different applications.

In one embodiment of the present invention, a view for a class, C, is implemented by listing functions of the class, C. A more flexible embodiment of the present invention allows the inclusion of composed functions with domain C in order to reproduce a full tree structure, such as tree 450, which includes composed functions such as the ISBN Number of the Content, or the Author of the Content, or the Name of the Author of the Content. The term "composed functions" refers to conventional function composition, wherein a second function whose domain is a sub-class of the co-domain of a first function is applied to the result of the first function.

In a preferred embodiment of the present invention Views are defined recursively as follows. Each View of a class includes a list of functions with domain C and, for each function, a view of its co-domain. Reference is now made to FIG. 4C, which is a simplified illustration of a View, in accordance with a preferred embodiment of the present invention. FIG. 4C shows a view for Books, which can be used to create a View for Libraries in FIG. 4B. I.e., a View for Libraries includes the function Contents, and uses the View in FIG. 4C for displaying the View of the content.

A typical ontology model uses standard abstract fundamental classes such as integers, characters and character strings, which are, in accordance with a preferred embodiment of the present invention, provided together with appropriate fundamental views. E.g., a fundamental class Integers may be provided with a View such as decimal representation View or Roman numeral View, which are implemented directly in software rather than being defined recursively in terms of other views. Such fundamental views are used to terminate recursive definitions of other views.

It may be appreciated by those skilled in the art that a view corresponds to an xsd:complexType element in an XML Schema.

Preferably, the distributed ontology of the present invention includes a model for authorship. Reasons for wanting to know about the author / publisher of a class or relation include:

- Is this a well known author (such as an industry association) whose definition is likely to be adopted as a standard?
- Is the definition likely to be clearly worded?
- Is the class / relation definition likely to be kept stable?
- Is the class / relation server likely to be kept reliably on-line?
- Where is the ontology physically distributed?

A preferred approach to authorship meta-data is that each class and relation definition has associated therewith

- The name of an entity who is responsible for authoring the document;
- A digital signature of the entity on the class / relation definition document, using one of the well known standards for digital signature such as the standards for digital signature on an XML document;
- A certificate or chain of certificates signed by a well known certification authority associating the digital signature with the author; and
- Other authorship or meta-data used in the art (e.g., as in the OIL model).

Preferably an author also signs a record of a URL in order to verify that he is responsible for the up-time of the URL. Otherwise, anyone could copy a signed document and present it from a non-reliable web server, which would not provide a suitable URL for referencing the class.

In a preferred embodiment, the present invention enables a user to list authors that he trusts. Moreover, in a preferred embodiment, the present invention supports user groups on the web maintained by a user group owner. A user can describe a "trust policy" by listing groups that he trusts. Preferably, groups are recursive, so that a group may contain another group by inclusion.

It may be appreciated that the filtering of ontologies based on a trust policy leads to the formation of sub-webs. A sub-web includes (i) class

definitions authored by approved authors; and (ii) relation definitions authored by approved authors and for which classes comprising the domains are authored by approved authors; (iii) function definitions authored by approved authors and for which classes comprising the domains and co-domains are authored by approved authors; and (iv) inheritance, where the subclass, superclass and inheritance are all by approved authors. Such an ontology sub-web is available to a user based upon his trust policy. The toolkit described hereinbelow ensures that only a user's trusted sub-web is transparent to him.

In a preferred embodiment, the present invention includes an authoring tool, which is a set of graphical tools for creating class and relation definitions. The authoring tool allows an ontology web to be viewed and new classes and relations added thereto.

In a preferred embodiment, the present invention includes an ontology web toolkit that provides interfaces to class and relation servers. Preferably, the toolkit includes an application programming interface (API) for finding classes and accessing them, including:

- Finding a class by specifying any of the following or combinations thereof:
  - Key words from definition
  - Superclass
  - Subclass
  - Author, or an entity along the certification chain
- Given a class, access methods for:
  - Superclasses
  - Subclasses
  - Relations with this domain
  - Definition

Preferably, the API includes similar interfaces for finding and accessing relations.

The present invention includes a global ontology directory, such as ontology directory 243 (FIG. 2). Preferably, new class and relation definitions have their URLs submitted to the global ontology directory in order to become part of the ontology web. In a preferred embodiment of the present invention, global ontology directory 243 maintains the following indices using a database table or search engine technology:

- A list of class definitions indexed by keywords in their descriptions;
- A list of class definitions indexed by superclass references;
- A list of class definitions indexed by subclass references;
- A list of relation definitions indexed by keywords in their descriptions;
- A list of relation definitions by each domain class reference;
- A list of function definitions indexed by keywords in their descriptions; and

- A list of function definitions by each domain and co-domain class reference.

In a preferred embodiment of the present invention, when listing class definitions indexed by superclass or subclass references, classes are included in such a list irrespective of whether or not the class declares the indexed superclass or subclass to be a superclass or subclass thereof, respectively; or whether or not the indexed superclass or subclass declare the class to be a subclass or superclass thereof, respectively.

In a preferred embodiment the present invention also includes an ontology software library, or toolkit, including an application programming interface (API) for application users of the ontology, enabling them to (i) locate published class and relation definitions; (ii) traverse a web of class and relation definitions, (iii) evaluate reliability of ontologies based on authorship and authorship of dependents, and (iv) negotiate common ontologies.

Such an API may be implemented directly by those skilled in the art by using remote procedure calls, such as remote method invocation (RMI), to call corresponding functions / methods on directories and servers, or by requesting ontological information from a relevant server, for example, by using standard HTTP, and then parsing it using, for example, a DOM or SAX interface XML parser to glean necessary information.

Recognizing that more than one player may provide alternative models for the same part of the real world, the present invention preferably includes an "equivalence" relation, similar to the inheritance relation, to identify classes which are equivalent. Two classes are equivalent if there is a one-to-one onto function with one class as its domain and the other class as its range. In particular, if two classes are equivalent, then instances of the first class are equivalent to instances of the second class.

Reference is now made to FIG. 5A, which shows an ontology network with class hierarchies from different domains and functions that relate between them, in accordance with a preferred embodiment of the present invention. Distribution allows the different domains to be stored and maintained separately, whether or not physically separated, while sill comprising a single ontology. Relations span classes from different domains that combine into one master ontology. Reference is now made to FIG. 5B, which shows how different domains may be authored by different companies or organizations who are authorities in their field, in accordance with a preferred embodiment of the present invention. Thus an ontology network may have many contributors.

Reference is now made to FIG. 5C, which shows how different authors may have different ownership models for sharing their models, in

accordance with a preferred embodiment of the present invention. Preferably, contributors choose the basis for sharing ontologies.

Reference is now made to FIG. 5D, which shows how some parts of the ontology may be commercially sensitive and not shared, in accordance with a preferred embodiment of the present invention. Intel, for example, may have private extensions to an ontology web that are invisible to the outside.

Reference is now made to FIG. 5E, which shows how a user filters domains based on trusted authorship, in accordance with a preferred embodiment of the present invention.

Reference is now made to FIG 5F, which shows how there may be equivalent classes, such as PCs, within an ontology, in accordance with a preferred embodiment of the present invention. Preferably, two-way inheritance or a function marked "equivalence" between them, which may be marked by constraints as 1-1 and onto, establishes the equivalence. Thus it may be appreciated that the web accommodates equivalent and overlapping ontology models.

Reference is now made to FIG. 5G, which shows how classes and functions are added according to safe transformations only, without disrupting existing uses of a model, in accordance with a preferred embodiment of the present invention. Thus it may be appreciated that the web can expand without compromising backward compatibility.

Reference is now made to FIG. 6, which shows conceptually that an ontology model acts as a master thesaurus/dictionary to which other data standards may be mapped, in accordance with a preferred embodiment of the present invention. Shown in FIG. 6 are (i) XML Schemas 610 (standards-based or proprietary); (ii) relational schemas 620 (for on-line or data warehouse type applications); (iii) data 630 accessed through an API, for example, of an enterprise application such as ERP or encapsulated in objects on the net such as CORBA, Java RMI or Microsoft DCOM; (iv) data 640 in older fixed-format standards such as EDI (EDIFACT or X.11) or FIX; and (v) data 650 accessed through web service protocols such as the SOAP, remote procedure call via XML standard.

Reference is now made to FIG. 7, which shows a relationship between analysis and run-time data management environments, in accordance with a preferred embodiment of the present invention. At analysis time an ontology model is preferably generated using an authoring platform directly by data modelers and/or by importing data schemas/formats in use in the relevant part of the enterprise.

The platform may derive transformations such as SQL queries or XSLT scripts for transforming records/instance documents between any pair of such data formats that are mapped to the ontology. The platform may also map beyond a pair, to aggregate data from two separate but related data sources.

These transformations may be executed periodically at runtime by standard Middleware, such as IBM MQ-Series or Tibco message-oriented middleware, or Enterprise Application Integration (EAI) solutions, such as IBM Websphere MQ Integrator, Webmethods, SeeBeyond, and BEA E-Link, which are responsible for scheduling and administrating execution of the transformations.

In a more sophisticated embodiment, the runtime environment may query the ontology model on-the-fly when presented with a data movement task and request a suitable transformation for the source and target databases or data formats specified.

Reference is now made to FIG. 8, which shows how a traditional EAI platform is replaced with a more flexible web approach utilizing an ontology of the present invention. Shown in FIG. 8 are data sources, such as a relational database management system, which are connected to an adaptor/server that publishes the existence of its data records by looking up how this data is represented in the ontology and transmitting this information to a central knowledge directory. For example, for each table the server may publish the details of

- the ontology class corresponding to that table; and
- a list of the functions of that class corresponding to the fields in the table.

A client application searching for information; for example, a function which is the property of some instance, can then use an adaptor/client implemented, for example, as a software library with an API, to search for data sources which contain instances of that class and which have such function as one of the fields they maintain. The client may check if the particular instance is present using either a global id, as described hereinbelow, or a given view with a combination of properties that uniquely identifies the instance.

The type of knowledge web environment provided by the present invention is superior to prior art EAI approaches in that it allows new sources of data in the enterprise, or across enterprises using the Internet, to be immediately utilized by client applications looking for each data. In distinction, in prior art EAI environments the link has to be configured manually for each relevant client or for groups of clients using publish/subscribe.

Reference is now made to FIG. 9, which is a simplified diagram of a gross profit calculation through an ontology model. Fig 9 shows how a

central ontological model can be used to indicate a relationship between disparate relational or XML data formats. FIG 9 also shows how a central ontological model can be used to ease performance of a query that involves more than one database.

Shown in FIG. 9 is a sales module 905 of an enterprise resource planning (ERP) system, such as SAP, including two tables: a sales table 910 with one line per sale, and a joined line items table 915 listing line items from the sale.

Table 920 is a table from a product data management system listing, for each of a plurality of SKUs, multiple rows of component numbers and quantities (not shown) giving the components part of a bill of materials for the SKU product.

Data store 925 is a stream of XML documents providing up-to-date prices for various components based on most recent transactions in an electronic business-to-business marketplace, such as cXML documents flowing between a marketplace and an Ariba procurement system.

In order to calculate current gross profit margin for a product it is necessary to look up its sales price in ERP sales module 905 and calculate its ratio to the cost of goods, which in turn requires looking up a list of components in the table 920 and then calculating the value of each component by searching for the most recent relevant component cost in data store 925. It is apparent that without a central model this would require manual research and mapping of the different data formats.

Central ontology 930 is an ontology model including three classes; namely, a Product class 935, a Component class 940 and a Currency class 945. FIG. 9 shows the relationship between them using the functions Price, Bill-of-Materials and Value. The four lines 950 show correspondences between a field in a relational database and a class in ontology 930. Two further lines 955 show the correspondence between a type in an XML Schema and a class in the ontology.

Central ontology 930 can also be used to transform data between different formats. For example, the gross profit information could be fed into a separate gross-profit database that is also mapped to the ontology or to a different format, such as an Excel spreadsheet. In order to do so the ontology is extended with a Gross-Profit-Margin function from Products to Ratios.

As described hereinabove, a third application of central ontology 950 is an instance browser that provides the ability to browse instance data in disparate systems in a consistent and seamless way. Using an instance browser, a user can look at a product instance, use a menu to view its price, use another

menu to view its components as icons, and use a menu of the component icon or a submenu of the product menu to view the current value of that component.

Implementation Details

Reference is now made to FIG. 10, which is a simplified block diagram of components of a comprehensive ontology system, in accordance with a preferred embodiment of the present invention. Shown in FIG. 10 is an ontology layer 1010. Ontologies are described hereinabove.

Also shown in FIG. 10 is a constraint layer 1020. Constraint layer 1020 is an extension to ontology layer 1010 that allows more advanced information about a model of the world, beyond just vocabulary, to be captured.

Typed first order logic can be used to provide a very general constraint model. Classes provide logical types and inheritance represents sub-typing. Relations are logical predicates, and functions are logical functions. Templates can make it easy to express common constraints such as:

- Two classes are disjoint.
- Two functions or a function and relation, are inverse to each other.
- A function is one-one (alternatively, onto).
- Limitations on the cardinality of a relation.

Given two relations R_1 and R_2, expressions such as $\sim R_1(a,b)$, $R_1(a,b) \wedge R_2(a,b)$, $R_1(a,b) \vee R_2(a,b)$, $(\forall x \in C)\ R_1(a,b,x)$ and $(\exists x \in C)\ R_2(a,b,x)$ can be used within logical formulae of typed first order logic. In these expressions, "a" and "b" are specific instances, and "x" is a variable instance from a class C. Similarly, collections of instances can be defined by logical expressions such as $\{x \in C: R_1(a,b,x)\}$. For example, a collection of books written by an person a can be defined as $\{x \in C: \text{Author}(x,a)\}$, where C is a class of books, and Author(x,a) indicates the relation that person a is an author of book x.

It may be appreciated by those skilled in the art that there are two layers of logic for an ontology. The first layer uses only variables, but no specific instances, and can be used to provide constraints on the ontology. The second layer uses instance constants, and can be used to make statements about specific instances and define sets of instances.

In accordance with a preferred embodiment of the present invention, ontology modelers can choose templates that are useful for describing rich information about their domains, rather than expressing themselves directly using first order logic. An important set of constraints are those that correspond to constraints that exist implicitly in the structure of relational databases. It is noted that two constraints, which directly impact vocabulary, are included already

in the ontology model; namely, inheritance and functions, which are constrained relations.

In a preferred embodiment of the present invention, version control layer 1030, as described hereinbelow, allows the evolution of a model to be tracked. Version control provides a general model, which is herein applied to ontology, for tracking changes to a model and avoiding clashes between changes. Version control allows the evolution of a model to be tracked. Preferably, layer 1030 provides a simple model of version control which can be integrated directly within ontology tools.

An alternative embodiment is to integrate external version control software but this embodiment has a disadvantage of storing models in many small files, in order to allow high resolution check-in and check-out, since external version control software controls versions at the file level.

In general, version control is useful for a project or model where:

- The model is separable into a number of components. In the case of an ontology model, the components are the "concepts;" namely, the classes, relations, functions and inheritances. In other cases, the components are operating system files.
- Multiple users are editing the model.

Version control has the following aims:

- Ensure that two edits do not clash, by ensuring that no two users are editing the same component at the same time, referred to as "pessimistic version control." In alternative embodiments, referred to as "optimistic version control," parallel edits are permitted and flagged.
- Allow temporary edits to be tested by the editor transparently to other users.
- Track changes
  - Maintain a complete history of every component.
  - Maintain a record of who made each change, his comments on the change and grouping of one change that affects multiple components.
  - Note particular points in time where all components of the content are given a version number.

Version control is preferably based on the following principles.

- Fine-grain version control at the level of components which coincides with concepts
- Pessimistic version control – no support for multiple users checking out the same component, which is necessary when fine-grained version control is in use.
- Linear versioning – no support for branching of versions at this control.

FIG. 10 also includes an access control layer 1040. Access control provides a general model, which is here applied to ontology, for managing the assignment of permissions in a system with many resources, many users and many types of resource access.

The basic concepts in access control are as follows.

- User – an individual who wishes to access resources in a system.
- Resource – a resource to which different users are given access. Each resource is preferably associated with a unique User known as the Owner, who is the ultimate authority on who can access the Resource.
- Resource Category – a category of Resources for which generally the same access types apply.
- Access – a specific type of operation such as read, write (or at a higher granularity, read a certain field) which is relevant to a specific Resource Category.
- Privilege – the granting to a User of a specific Access right to a specific Resource.

In principle access control can allow each Resource Owner to list, for each of his Resources and for each User, a list of Access privileges permitted to that User for that Resource. However, this quickly becomes unmanageable. To mitigate the management difficulty, the following concepts are introduced.

- User Group – a set of Users, or of other User Groups, where this set is determined from time to time by a User Group Owner.

  A User Group is preferably used to group together a certain set of Users (e.g., "Gold subscribers," "IBM employees," "IBM Finance Department employees," "users with no fines in the last two years") with some common property or who are to be entitled to some common set of privileges. The User Group Owner will be someone who is trusted, by at least some owners, to manage the User Group reliably, so that the Owner will feel confident in assigning privileges to the whole group.

  Preferably, recursive use of User Groups is permitted; i.e., inclusion of group A as a member of group B means that all members of A necessarily satisfy the criteria to make them members of group B. E.g., "IBM employees" may have members such as "IBM Finance Department employees." When assigning a privilege to group B one is by implication assigning privileges to members of group A, and therefore trusting that the Owner of Group B, in assigning Group A as a member, is also monitoring who the Owner of Group A is admitting as members.

The term User Grouping is used for either a User or a User Group.

- Resource Group – a set of Resources, or of other Resource Groups, all of which are of the same Category and have the same Owner, who is also the Owner of the Group and who determines which Resources are in the set.

A Resource Group is preferably used to group together a certain set of an Owner's Resources to which the Owner may want to assign privileges at some time (e.g., "resources I share," "most sensitive resources," "resources I share with ABC").

Preferably, recursive use of Resource Groups is permitted; i.e., inclusion of group A as a member of group B means that all members of A necessarily satisfy the criteria to make them members of group B. E.g., "Resources I share with ABC" may have members such as "Resources I don't care about".

The term Resource Grouping is used for either a Resource or a Resource Group.

- Access Group – a set of accesses, all relevant to the same Group. These may be defined recursively and are pre-defined for any given system.

The Access Group is preferably used to define access types that typically go together as a level of privilege. E.g., "Write Privileges" may include the Access "Write field 1," "Write field 2" and the Group "Read Privileges."

- Access Control List – a unique set attached by a Resource Group Owner to a Resource Grouping listing multiple User Groupings and, for each User Grouping, a list of Access Groups indicating privileges which the User Grouping is granted to Resources in the Resource Grouping.

Access control layer 1040 preferably allows anyone to create and manage a User Group. It further allows any Resource Owner to create and manage Resource Groups. It further allows any Resource Owner to attach and manage an Access Control List for each Resource Grouping.

A User is granted an Access to a Resource if such Access is in at least one Access Grouping granted to at least one User Grouping in which such user is a member, for at least one Resource Grouping in which such Resource is a member.

Preferably only a Resource or Resource Group Owner can change ownership of a Resource Grouping. A change of ownership for a Resource Grouping preferably recursively changes ownership of member resources, and removes the Resource Grouping from Resource Groups to which it belongs.

Control of creation of Resources is preferably exercised by associating a command to create a Resource, with another Resource. For example, in a file system a directory may be a Resource and the Resource Owner decides who has permission to use the directory's create file Access, to create a new file in the directory owned by its creator.

Thus, using access control an author of a class protects who can see the class, who can read it, who can update it (i.e., make changes of a permitted type) and who can edit it. Access can be controlled at the fine resolution of individual classes, relations, functions and inheritance pair definitions, each of which are resources that have an owner who assigns access permissions to others.

FIG. 10 also includes a relational sub-ontology layer 1050. Relational sub-ontologies are subsets of an ontology that correspond to a specific relational database.

FIG. 10 also includes a relational mapping layer 1060. A relational mapping is an actual correspondence between a suitable subset of an ontology and a specific relational schema. Specifically, each field type in a relational database may correspond to a class in the ontology. Each n-field table in a relational database may correspond either to an n-ary relation on the n classes corresponding to the field types, or to a class. In the latter case, each field type induces a binary relation between the class corresponding to the table and the class corresponding to the field type.

Reference is now made to Table I, which is a simple example of a relational database table.

Table I: Flight Bookings

Person	Flight	Seat
John	001	10B
Fred	002	20K

The field types Person, Flight and Seat, which may in themselves be joins to other tables, correspond to classes Persons, Flights and Seats in an ontology. The table Flight Bookings may be considered as a tertiary relation with domain Persons x Flights x Seats. This relation includes two triplets, corresponding to the two rows in Table I.

Alternatively, the table Flight Bookings may be considered as a class Flight Bookings, with two instances, corresponding to the two rows in Table I. The field type Person induces a binary relation between Flight Bookings and Persons, in which the first Flight Bookings instance is related to Persons instance

John, and the second Flight Bookings instance is related to Persons instance Fred. Similarly, the field type Flight induces a binary relation between Flight Bookings and Flights.

It may be appreciated that if the table Flight Bookings is used as a type in other tables, it is preferable that the table correspond to a dedicated class, rather than to a relation.

FIG. 10 also includes an instance structures layer 1070. Layer 1070 preferably provides the ability to directly document instances according to an ontology.

Preferably, an instance structure associates

- A set of instances B(C) with each class symbol, C, such that the set associated with a subclass is a subset of the set associated with a superclass thereof.
- A relation on (i.e. subset of) $B(C_1) \times B(C_2) \times \cdots \times B(C_n)$, with each relation symbol R of domain $C_1 \times C_2 \times \cdots \times C_n$.
- A function from B(S) to B(T) with each function symbol f of domain S and co-domain T; i.e., for each instance $x \in B(S)$ there is associated precisely one element $f(x) \in B(T)$.

The concept of an instance structure contributes to understanding an ontology, since an ontology is purely a guide for creating instance structures and a vocabulary for talking about them.

FIG. 10 also includes a view layer 1080. As described hereinabove, Views are preferably used to provide templates for talking about instances by documenting specific combinations of attributes that define or reference an instance in particular circumstances. Views are an extension to the ontology model. One or more Views may be attached to each class in an Ontology Model.

As well as defining Views abstractly, Views may be generated for specific instances from the instance structure described hereinabove.

FIG. 10 also includes a View / XML-Schema mapping layer 1090. The abstract model of a View, defining a tree of attributes, is similar to the syntactical schema of an XML Schema document. By mapping an equivalence between a View and an XML Schema document, the XML documents validated by the Schema may then be used to describe instances. Such a mapping is preferably created between an existing Ontology and XML Schema, or alternatively, the XML Schema may be generated on the fly together with the mapping. The mapping associates Views with XML Schema types and adds syntactic information such as element names.

In a preferred embodiment of the present invention, Packages are used to manage the many names that can arise in an ontology web. Definitions are

sorted into Packages, an idea borrowed from Java. Preferably, Packages are implemented as a simple hierarchy starting with a Root Package which has no parent. Every Concept (class, relation, function, inheritance) preferably belongs to one Package, and its full name is preferably the Package path from the root plus the Concept name.

Names of a Concept are preferably unique within a Package. When defining a Concept within one Package, a GUI preferably recognizes local names within that Package without a path. Other Packages are imported for convenience, provided there are no name conflicts. Preferably, Packages have owners who control who can add or delete things therefrom. Each Package and each Concept is a Resource. Creating a new Concept is preferably done through a Package, subject to permission to create.

Packages are a Resource type. Their Accesses are API calls that are grouped into

- Create sub-Package
- Delete sub-Package
- Delete Package
- Create Concept.

Concepts are a Resource type. Their accesses are API calls that are grouped into

- See (i.e., see that it exists)
- Read (includes See)
- Update (includes Read plus check-out and certain edits) – perform changes that preserve backward compatibility only
- Write (includes Updates)
- All (includes Write plus Delete)

Preferably, each Resource is also a component for version control purposes.

FIGS. 11 – 17 are Unified Modeling Language (UML) static structure class diagrams, which show how to implement the various models described above using an object oriented programming language such as Java or C++, in a preferred embodiment of the present invention.

Reference is mow made to FIG. 11, which is a UML diagram of a preferred version control model for version control layer 1030 (FIG. 10). FIG. 11 shows an abstract approach to version control. Preferably the ontology platform applies this model with Content corresponding to Model, and Component corresponding to Concept. As mentioned hereinabove, Concept preferably includes classes, relations, functions and inheritances.

As can be seen in FIG. 11, each component of the Content is preferably abstract and long-lived, and corresponds to a sequence of ComponentVersions, each of which contains the content of the Component at a subsequent period of time. The last ComponentVersion is the "current" version. A component that is deleted is associated with an EmptyComponentVersion as its current version.

A change to the model is preferably achieved by a user creating a Checkout, which may include new temporary CheckedOutComponent versions of all the components that have to change in this unit of work. Preferably, a component may only be checked out once at any time in one Checkout. A component is deleted in a CheckOut using the EmptyCheckedOutComponent.

To the extent possible users are encouraged to include all Components that will be included in a Checkout from the beginning, in order to avoid these changing during the unit of work and to avoid a deadlock where two users are waiting for each other to finish. To the extent that the language allows it, the CheckOut is preferably expanded to include a set of Components that may be changed freely without depending on other Components for consistency.

In general a user should base CheckedOutComponents on the latest ComponentVersions, although this is not obligatory. Occasionally a user may want to base a new ComponentVersion on an historical version, which is permitted provided that at check-in time the new check-in creates a consistent model, and an extension of the current model if this is enforced.

At check-in time the Checkout is preferably destroyed and the CheckedOutComponents becomes the newest ComponentVersions for the relevant Components. Note that a check-in preferably must be tested for consistency and added to the model in one transaction to avoid inconsistencies.

Every check-in represents a new ContentVersion. The ContentVersion can be used to access a consistent set content by ensuring that all updates, i.e., check-ins, up to a certain point are viewed. Optionally the ContentVersion may be given a version number representing a release.

Every ComponentVersion is part of one or more ContentVersions; namely the ContentVersion created by the check-in that creates such ComponentVersion and every ContentVersion subsequent thereto up to the next check-in creating a version of the same Component.

Preferably, when editing a user may view any ContentVersion, although typically the most up-to-date one available will be edited. The user may additionally, subject to access control, apply deltas; i.e., changes, of some checkouts, in order to test them prior to check-in.

The version control model as described thus far allows changing a component by creating a new version. In drafting stages there may be many insignificant stages in which storing new versions is unnecessary. In this case a component preferably may be marked draft, which limits its publication, and new versions when checked-in replace the most current version.

The restriction that every concept is associated with at most one CheckedOutComponent may be relaxed, giving an optimistic locking scheme. Assuming the components are fine-grained, as in the case of concepts, no merging is offered. Instead, if two parties change a component simultaneously, the second check-in will either fail since it will be a non-allowed expansion to the model, or succeed and overwrite the first change. However, if expansion is enforced then the second check-in is an expansion of the first, so the first will not be lost.

Optimistic locking has the disadvantage of possible overlaps of work causing a loss of work. It also has substantial scalability advantages:

- No limitations on the ability to check-out.
- Check-outs need not be registered in any way with central database until check-in time.

Reference is now made to FIG. 12, which is a UML diagram of a preferred access control model for access control layer 1040 (FIG. 10).

Reference is now made to FIG. 13, which is a class diagram for an ontology model, in accordance with a preferred embodiment of the present invention.

In a preferred embodiment of the present invention, a relation and a class are both types of statements. This is apparent for a relation, but additionally a class Book forms a statement "is a book", which is always unary. Preferably, inheritance is modeled with a helper class in order to allow separate authorship of inheritance relations. Concepts are the primary components of an ontology, which preferably have a label and description in order to associate them to the real world. Preferably, Being does not have superclasses. Further types of complex classes, such as intersections, sets, lists and sequences, are not shown in FIG. 13.

The bottom of the diagram in FIG. 13 shows how an access control model is implemented in a preferred embodiment, as described hereinabove.

Reference is now made to FIG. 14, which is an extension to FIG. 13 accommodating instance structures, in accordance with a preferred embodiment of the present invention. Instances are illustrated as being associated with one class, and it is implicitly implied that they also belong to superclasses thereof. Preferably, there is one FunctionValue object for each instance and for

each function defined on the class of that instance or on any superclass of that class. The RelationValue preferably contains a tuple of instances suited to the domain of the relation, or to subclasses thereof.

Reference is now made to FIG. 15, which is an extension to FIG. 8, showing how concepts may be organized according to a hierarchy of packages in accordance with a preferred embodiment of the present invention.

Reference is now made to FIG. 16, which is an extension to FIGS. 13 and 15, showing which objects are treated as resources for the purpose of having independent access control in accordance with a preferred embodiment of the present invention.

Reference is now made to FIG. 17, which is a UML model for views and their relationship to ontology models. As can be seen in FIG. 17, there are several types of views, including fundamental views, list views, set views and customized views. Fundamental views are preferably used for viewing fundamental classes, such as integers and text strings. For example, fundamental views of integers may include decimal representation, hexadecimal representation and binary representation.

Set views are preferably used for viewing sets of instances of a class, and list views are preferably used for viewing lists of specific instances of a class. Customized views are composed of view elements. Set views, list views and customized views are described hereinbelow with reference to FIG. 31A.

Reference is now made to FIG. 18, which is a simplified block diagram of a run-time platform for implementing a preferred embodiment of the present invention. Preferably, the platforms cater to collaborative authoring and managing of all aspects of an ontology model, and transform instance-data between data sources.

In a preferred embodiment of the present invention, a Java model is created in parallel to the ontology model. The Java model is described hereinabove by the UML dagrams in FIGS. 11 – 17. Preferably, the architecture used to implement the present invention enables persistence and simultaneous access.

A single master copy of the ontology model, including versioning information and history and access control parameters, is stored in a database 1810. A model encapsulation layer 1820, such as EJB Entity Beans, provides scalable convenient access by keeping potentially multiple copies in RAM and providing access to them through an API that has methods appropriate to the model. Access control module 1830 reads access control data from the database and enforces access control. A model access logic layer 1840 implemented, for example using EJB Session Beans, preferably provides higher

level functionality, such as searches or other manipulations, which break down into several atomic accesses to the model. Model access logic layer 1840 has its own access control 1850.

The reason for two access control layers is as follows. Suppose a user wants to delete all classes with a name starting with 'z'. One may wish to block certain users from doing such dangerous activities using the right hand access control 1850. On the other hand, even if a user is allowed access to this method one would then wish to ensure at the underlying layer that this does not result in the deletion of specific classes which this user is not allowed to delete. Thus the authenticated user identity is propagated by the sessions to their access to the model encapsulation and privileges are verified again for each underlying action.

Finally the user interface layer shows some options for user interface such as a web interface 1860, an applet 1870 or a standalone application 1880, all of which may connect to the model access using, for example, Java Server Pages for web interface 1860, or Java Remote Method Invocation (RMI) for applet interface 1870 or standalone application interface 1880.

Instance Browser

In a preferred embodiment, the present invention uses a graphical user interface to provide visual graphical elements which correspond directly to classes, instances, relation values and statements on the ontology web, or statements on a knowledge web that uses vocabulary of the ontology web to introduce instances and statements about instances.

Reference is now made to FIGS. 19A – 19F, which are illustrations of the use of icons for browsing instances that are organized according to an ontology, and logically defined sets of such instances, in accordance with a preferred embodiment of the present invention. FIG. 19A indicates an icon for a class of Sports Utility Vehicles. FIG. 19B indicates an icon for a specific instance of a sports utility vehicles. Preferably the icon in FIG. 19A, which represents an abstract class, is abstract, and the icon in FIG. 19B, which represents a specific instance, is photo-realistic.

FIG. 19C indicates an icon for an attribute, or property, of a class. FIG. 19D indicates an icon for accessing all attributes of a class.

FIG. 19E indicates an icon for a collection of books. FIG. 19F illustrates the collection of books corresponding to the icon of FIG. 19E. That is, the icon of FIG. 19E illustrates a collection of books that is iconized by a user, and FIG. 19F illustrates a collection that has been opened. It is noted in FIG. 19F

that the collection is referred to as Restaurant Cooking, and the specific books in the collection are indicated by icons 1910 for instances of books.

Reference is now made to FIGS. 20A and 20B, which are illustrations of processes associated with instances of classes, in accordance with a preferred embodiment of the present invention. FIG. 20A indicates an icon used to represent a process. FIG. 20B illustrates cascaded menus for processes associated with an instance of a book from the collection of Restaurant Cooking Books (FIG. 19F). The processes include (i) buy a similar book (2010), (ii) contact the author (2020), (iii) contact the publisher (2030), (iv) join a discussion board (2040), (v) sell your copy (2050), and (vi) submit your recipe (2060). The icon of FIG. 20A appears adjacent to each of the processes 2010 – 2060.

Reference is now made to FIGS. 21A and 21B, which are illustrations of a page containing links to various classes, in accordance with a preferred embodiment of the present invention. A page is a generalization of an HTML page with the additional capability of embedding icons representing classes, instances and collections that are relevant to contents of the page. FIG. 21A indicates an icon used to represent a page. FIG. 21B illustrates a page for a book entitled "Top Secret Restaurant Recipes." The page includes (i) a process icon 2110 for accessing processes associated with the illustrated book, such as the processes shown in FIG. 20B; (ii) an attribute icon 2120 (cf. FIG. 20C), for requesting information about the attributes of the illustrated book; (iii) an icon for a collection of books 2130 (cf. FIG. 20E) entitled "Restaurant Cooking" and (iv) an icon for a collection of books 2140 (cf. FIG. 20E) entitled "Exotic Cookery." Both collections are related to the illustrated book.

Reference is now made to FIGS. 22A – 22E, which are illustrations of an instance browser for navigating through class and relation definitions, in accordance with a preferred embodiment of the present invention. Shown in FIG. 22A are windows containing instances of classes, and collections of instances of classes. Individual instances are depicted by photo-realistic icons, and collections of instances are depicted by folders. A window 2210 contains folder icons 2211, 2212 and 2213 for collections of instances of suppliers, subcontractors and carriers, respectively. Window 2220 contains instance icons 2221, 2222, 2223, 2224, 2225 and 2226 for departmental staff.

Shown in FIG. 22B is a window 2230 with instance icons 2231, 2232, 2233 and 2234 for carriers. Preferably, window 2230 pops up when a user right-clicks on folder icon 2213.

Shown in FIG. 22C is a list 2240 of options relating to a specific carrier, UPS. List 2240 includes an item 2241 for properties of the UPS carrier,

an item 2242 for processes for the UPS carrier, an item 2243 for help, an item 2244 for search and an item 2245 for view page.

Shown in FIG. 17D is a further list 2250 of properties for carrier UPS. Preferably, list 2250 pops up when a user clicks on item 2241 in list 2240 (FIG. 22C), requesting to view properties of the UPS carrier. As seen in FIG. 22D, displayed properties include name 2251, location 2252, employees 2253, credit rating 2254, volume 2255 and rates 2256.

Shown in FIG. 22E is a list 2260 of UPS locations. Preferably, list 2260 pops up when a user clicks on item 2252 in list 2250 (FIG 22D), requesting to view locations of the UPS carrier. Each location in list 2260 itself an instance of a class for locations, and can be further identified by clicking on it. It is thus apparent that the user interface illustrated in FIGS. 22A – 22E can be continued recursively.

It may be appreciated that what is presented to the user as an attribute of a class is the value of a function appropriate to the class.

An advantage of the user interface shown in FIGS. 22A – 22E is that various information about UPS from different sources, including procurement managers, supplier management systems, Dunn and Bradstreet, and UPS themselves, is presented through a right-click on the UPS icon, since the icon represents an agreed upon ontological reference to UPS. Upon right-clicking on the icon, a graphical user interface tool preferably searches for statements about UPS using an agreed upon ontological vocabulary; namely, functions that have the class Companies as domain. Specifically, the graphical user interface tool searches the ontology for attributes of the class of the instance, and then searches for instance information, including values for these attributes, as described hereinbelow.

Ontology allows values themselves to be instances, so that a user can navigate through the knowledge. For example, after finding a location of UPS, the user can then look for information on how to get to that location. Similarly after finding the author of a book, the user can right-click on an icon of the author to find the author's phone number, or other publications by the same author.

In order to track data on instances, a preferred embodiment of the present invention defines within an ontology web a function called GID (for global identification) from Being to the natural numbers. As mentioned hereinabove, "Being" is a superclass of all classes. If a constraint language is available, this function is preferably constrained to be one-one.

Information about instances is preferably published in the following way: a structured document, such as an XML document, lists the

instance's class, its GID, and one or more pairs of function name & value. Alternatively a View may be published in which a specific combination of function name & values is listed, and possibly composed functions as well. The value may be listed simply through its GID, or iteratively a View of the value may be used. Such documents are preferably published and indexed by a standard database or web search engine based on GID, and preferably also by class and by function names present.

An example of such a structured document is as follows:

```
<instance>
        <class>
                http://www.abc.org/ontology/cars/audi#A4
        </class>
        <instance>
                0947320494365087
        </instance>
        <value-pair>
                <name> manufacturer </name>
                <value> 234812942734 </value>
        </value-pair>
        <value-pair>
                <name> metal type </name>
                <value> 2348573209 </value>
        </value-pair>
</instance>
```

In a preferred embodiment of the present invention icons for displaying a graphical representation of an instance are introduced through a function called Icon that is defined from Being to Images, or to a more specific class such as Jpeg Icon Images.

In an alternative embodiment of the present invention, images are introduced only for classes, so that every instance in a class has the same icon with a different caption. Preferably, in every <class> declaration an icon field is added with a URL of an icon. Equivalently, there is a Caption function from Being to String, and if a constraint language is available it may be noted that Caption and Name coincide as functions on People.

In a preferred embodiment of the present invention, captions are preferably taken from a certain parameter of type String. E.g., in the declaration of a class People, a function Name from People to String is marked as the one used to generate captions.

In an alternative embodiment of the present invention, captions are implemented through a function from Being to Strings.

Preferably, there is more than one function for Icon and Caption, and each specific implementation of a browser tool uses a different one, or allows a user to select one.

In order to display an icon for an instance on a computer display; that is, an icon which looks similar to a Windows icon for a file but which instead represents some physical or real world entity, the following steps are preferably taken:

- The instance is stored using its GID, or alternatively according to a View for the class. E.g., a book is stored using Author, Title and ISBN View.
- The icon is looked up by searching for a document for the instance that also lists the icon URL property, or alternatively the icon is looked up by class.
- The caption is looked up by searching for a document for the instances that include the caption, or an appropriate other function like Name, for the instance.
- An icon and caption are created on the screen. It may be appreciated by those skilled in the art that standard library calls are used to create such icons; e.g., using Microsoft Windows libraries or Java Swing APIs.
- A pop-up menu is attached to the icon using the standard library calls so that it is displayed upon right clicking on the icon.
- The class of the icon is looked up directly through the relevant ontology server or via the ontology directory. For each function on the class, an appropriate item is added to the pop-up menu, such as Author or Name. If it is desired to display things other than attributes in the menus, the attributes are preferably grouped in a sub-menu under "attributes" within a primary pop-up menu.
- To each such attribute an action is preferably attached, and an icon for the attribute value is preferably created as follows. When clicking on an attribute the program searches for instance documents for the instance that include the relevant function. Additionally, if author filtering is in place, documents with an inappropriate author are ignored. The GID of the instance is retrieved and an icon created for it as above. For certain fundamental classes such as String a value is preferably displayed within the menu rather than creating an icon. For example, a name of a person is displayed as name=Fred in the menu, rather than clicking on Name to get a new icon for the string "Fred."

It should be noted that what is described here is not necessarily a standalone application, but rather a procedure for use by an application that needs to display instances. The advantages of this procedure are:

- Instances and their attributes are displayed consistently by all applications.
- When one application displays an instance, a user can seamlessly obtain attribute information on the instance from other applications.

Icons are preferably grouped together in windows, as is done in the familiar Windows interface to file systems. Grouping is preferably implemented by keeping documents representing the icons within a folder on the file system.

Window type user interface elements are preferably created for grouping icons by a logically defined property. Such elements are referred to as "collections." Specifically, a collection is a logically defined set of instances; i.e., instances which share a common property visualized by displaying their icons within a single window.

For example, within a procurement management system collections may have a yellow title, showing the logical definition of the collection in English. The staff of a department is defined by a relation, and the information looked up dynamically from an HR system so that appropriate icons are displayed in a window representing this collection. Similarly for stock, which represents the relation of content between StockRooms and Components, and where information ultimately comes from a company's ERP system.

Many types of collections are defined in terms of first order logic using an ontology as a types system, and many useful collections are defined with the use of relations. For example, for a relation WorksIn on domain Departments x People, one can define a useful collection of everyone who is related by WorksIn to a certain department instance. Similarly, for a relation Contains on StockRooms x Components, one can define a useful collection of component instances related by Contains to a certain stock room instance.

Systems such as an ERP system which have information on the relations between instances preferably publish in a format containing (i) the ontological name of the relation, and (ii) lists of appropriately typed tuples of instances. Such a format is closely related to the format in which relations are stored in a relational database, where each table represents a relation by a list of tuples.

An example of values for the above mentioned binary relation WorksIn are as follows:

<relation-values>

<relation>

http://www.a-company.com/abc/human-resources/relations#worksIn

```
</relation>
<values>
                    <tuple>
                      <instance> 087434958743 </instance>
<instance> 598723459713 </instance>
                    <tuple>
                    <tuple>
                      <instance> 348534598234 </instance>
<instance> 349857632459 </instance>
                    <tuple>
            <values>
</relation-values>
```

Relations are preferably indexed using a standard database or web search engine based on the relation URL, and preferably also on the instances referenced, and preferably published using a standard web server. Information on author, or source of the information, may be added and used to filter trusted information.

In order to create a collection the following steps are preferably taken.

- A structure file, such as an XML file, stores a definition of the collection: a name of a relation and an instance for each class in the domain except one. The collection is defined to include instances that can fit in that missing class and satisfy the relation.
- A standard graphical element for displaying sets of things such as a window is instantiated from a standard library, such as a Windows or Java Swing GUI library.
- The caption for the collection is derived in English from the formal definition; e.g., "All People who work in finance". Here People is the name of the second class which is in the domain of the relation WorksIn. The separate words "works in" preferably come from a description field in a document declaring the relation. Finance is the caption of this instance of departments. Preferably this definition is displayed prominently to emphasize that the collection is not a file system folder.
- The web is searched for relation instance documents for the relation, and these are further searched for tuples where the second element is the instance finance. For each such instance an icon is created using the above mentioned technique, and inserted within the collection using standard library calls to add an icon to a window.

- The content of the collection is preferably refreshed periodically or upon demand as desired.

It should be noted that what is described here is not necessarily a standalone application, but rather a procedure for use by an application that needs to display sets of instances. The advantages of this procedure are:

- Collections of instances and their definition are displayed consistently by all applications.
- When one application displays an instance collection, a user can seamlessly obtain information on this instance from other applications. For example, an engineer browsing a stock room collection, ultimately coming from an ERP system, may right click on an item in stock to find technical information originating from the manufacturer.

The contents of a collection may be too numerous to be conveniently displayed in one window. Accordingly, the definition of the collection preferably lists a number of subclasses of the class of the instances; such as subclasses of components, and all instances falling under these subclasses are collected inside the collection window under an icon representing the class. Such instances are displayed in a subsidiary collection window, after double clicking on the icon.

Reference is now made to FIG. 23, which is a simplified block diagram of an architecture for an instance browser, such as the instance browser illustrated in FIGS. 22A – 22E, in accordance with a preferred embodiment of the present invention. As shown in FIG. 23, ontology server computers 2300 contain repositories 2305 of ontological information, such as class, relation and functions, as well as optional logical constraints. Ontology server computers 2300 also contain server components 2310 for responding for queries to the repository, and an optional publisher 2315 for notifying a central ontology computer 2320 of the existence and content of the repository. Cross-references between the different ontology repositories turn them into a single virtual ontological model.

Central ontology computer 2320 maintains a central index through which search engines may find ontological information; e.g. find all the functions which have a given class as domain.

Information about instances resides in one or more relational database table 2325 and or XML document repositories 2330. Data server adaptors 2335 are responsible for publishing the existence of the instance data using agreed ontological terminology and for responding to queries for specific instance data.

In order to achieve this, repositories 2340 and 2345 of ontology to relational and XML Schema mappings, respectively, preferably reside on data server-adaptors 2335 as shown. In an alternative embodiment of the present invention, repositories 2340 and 2345 may reside on ontology-server-computers 2300, or elsewhere. Preferably the data modeler uses an authoring tool to model the correspondence between tables and fields or XML Schemata and the types therein to the ontological model in the repository of class and relation definitions.

Publishers 2350 in data server adaptors 2335 use these mappings to create storages 2355 and 2360 of ontological information describing a specific relational table or document repository that they serve, using the agreed vocabulary from the ontological repositories. This information includes.

- The ontological class to which the table or schema correspond
- The functions whose values are given by the fields or elements therein.
- Optionally a logical predicate using typed first order logic as described elsewhere giving a constraint on the instances included. For example, a database only includes stock in North American stock rooms or only includes items priced below $100. It may be appreciated that such constraints save unnecessary searching. Preferably there are two predicates, one giving an upper bound on instances present so that there is no point searching for instances which do not satisfy that predicate, and one giving a lower bound so that the data source is guaranteed to include all instances satisfying the predicate. These predicates must typically be created by guidance of the database administrator.

A server 2365 in the data server adaptor is responsible for responding to queries that may have the form of a first-order logic term, or more simply a list of inequalities on different function values to specify the instances of interest. In an alternative but less scalable architecture the publisher actually uploads all the instances to the central instance computer, and no server is required. Server 2365 can use mapping 2340 in an obvious way to transform constraints on functions to constraints on corresponding fields, and to then obtain the relevant instances using an appropriate SQL statement or, for example,. an XQL statement in the case of XML documents. The instances are preferably transmitted by the server using, for example, an instance document as described hereinbelow.

A central instances computer 2370 maintains an index aggregating ontological info from different data sources. It allows quick searches for all data sources of instances of a given class with given functions of that class as domain. It may potentially be more sophisticated and further store predicates that provide more detailed information about the instances present.

An instance search engine 2375 searches for instances of a given class and with given functions required, or more generally for instances satisfying a first order logical predicate. Instance search engine 2375 first accesses central instances computer 2370 to compile a list of relevant data sources and then queries them directly.

Optionally instance search engine 2375 uses an inference engine 2380 so as to optimize the search. For example, given a potential data source, if the upper-bound predicate of that data source in conjunction with the search query is false, that data source may be ignored.

An instance browser 2385 is an application or applet which displays icons, collections and menus as in FIGS. 22A – 22E. It preferably uses standard Java Swing classes to realize the icons, windows and pop-up menus required. Once a collection has been defined instance browser 2385 preferably searches for relevant instances using search engine 2375. Instance browser 2385 may search one or regularly, preferably at the choice of a user. Icons may be assigned using a standard icon function of Being and searched for like other instances, or may be kept in a separate icon store 2390.

A user data store 2395 is preferably realized on hard disk by serializing the Java Swing classes or using an XML document or other standard programming technique familiar from window-like operating systems. User data store 2395 preferably includes:

- User options such as colors for a graphical user interface and frequency of updating collection information.
- Folders like operating system directories in which a user can collect instance icons
- Definition of collections defined logically by a user
- The state of a user's screen so that they may continue where they left off in the next session.

Ontology Authoring Tool

Reference is now made to FIGS. 24A – 24D, which are illustrations of an ontology authoring tool for creating an ontology model, in accordance with a preferred embodiment of the present invention. FIG. 24A includes a screen 2410 for creating a new ontology model. Screen 2410 contains two tabs: a tab 2420 for Version and a tab 2430 for Model. Screen 2410 corresponds to tab 2430, and contains text boxes 2440 and 2450 for entering a model name and a model description, respectively.

FIG. 24B includes a screen 2460 corresponding to tab 2420. Screen 2460 includes text boxes 2470, 2480 and 2490 for entering a version

author, a version label and a version comment, respectively. FIG. 24C shows screen 2460 with John Smith indicated as author in text box 2470, base classes indicated as label in text box 2480, and a comment included in text box 2490. FIG. 24D shows screen 2410 with IT ontology indicated as name in text box 2440, and a description entered in text box 2450.

Reference is now made to FIGS. 25A – 25D, which are illustrations of creating classes, properties, functions and inheritance within an ontology model, in accordance with a preferred embodiment of the present invention. FIG. 25A shows a screen 2505 for creating a new class within an ontology model. Screen 2510 includes text boxes 2510 and 2515 for entering a class name and namespace, respectively. A namespace, such as Newspaper.Journalism, is similar to the package naming convention in Java.

FIG. 25B shows a screen 2520 for creating a new property within an ontology model. Screen 2520 includes text boxes 2525, 2530, 2535 and 2540 for entering a property name, namespace, domain and description, respectively. Screen 2520 shows that Written_By is a relation on Article × Author.

FIG. 25C shows a screen 2545 for creating a new function, or property, within an ontology model. Screen 2545 includes text boxes 2550, 2555 and 2560 for entering a property name, and indicating whether the co-domain of the property is a simple or complex type class, respectively. A simple type class is one that exists within an ontology model, and a complex type class is one that is built of from existing classes by operations such as cross product, intersection, union, **set**, **bag** and **list**. Screen 2545 also shows that the domain of the property Manufacturer is a class named Servers, and its co-domain is a class named Company.

FIG. 25D shows a screen 2565 with a text box 2570 for defining a complex class. Shown in text box 2570 is a complex class Editor × **list**(Columnist × Article), which corresponds to a cross product of an Editor and a list of Columnist-Article pairs..

FIG. 25E shows a screen 2575 with an explorer-type class viewer, for adding a superclass.

Reference is now made to FIGS. 26A – 26F, which are illustrations of a class viewer with tabs for viewing class information, in accordance with a preferred embodiment of the present invention. FIG. 26A shows a screen 2604 having a left panel 2608 with an explorer-type viewer for classes, and a right panel 2612 for viewing general information about a class. Screen 2604 includes seven tabs 2616, 2620, 2624, 2628, 2632, 2636 and 2640 for viewing general information, properties, inherited properties, subclasses, enumerated values, relations and views, respectively. Screen 2604 corresponds to

tab 2616. Screen 2604 includes a box 2644 for listing direct superclasses, and a text box 2648 including a description of the class. As can be seen in panel 2608, a class Editor is selected. Box 2644 indicates that Editor is defined within a namespace Newspaper.Journalism, and is a subclass of Employees. Text box 2648 provides a description of the Editor class.

FIG. 26B shows a screen 2652 corresponding to tab 2620. Screen 2652 includes a box 2656 indicating that Sections is a property of Editor, with co-domain Section that is defined in a namespace Newspaper.Business. FIG. 26C shows a screen 2660 corresponding to tab 2624. Screen 2660 includes a text box 2664 indicating that Editor inherits properties By_Name, Current_Job_Title, Date_Hired and Salary from the superclass Employee; and inherits properties Name, Other_Information and Phone_Number from the superclass Person.

FIG. 26D shows a screen 2668 corresponding to tab 2628. Screen 2668 includes a box 2672 listing subclasses of Editor, of which there are none in this example.

FIG. 26E shows a screen 2676 corresponding to tab 2632. Screen 2676 includes a box 2680 listing enumerated values for a class named Weekly_Editions. Enumeration is used to define certain classes that can only contain a fixed finite number of instances. Screen 2676 shows a user adding an enumerated value, Sunday, to the list in text box 2680.

FIG. 26F shows a screen 2684 corresponding to tab 2636. Screen 2684 includes a text box listing relations that apply to Reporter. Screen 2684 shows that Written_By is a relation on Article × Author. It is noted that Reporter inherits the relation Written_By from Author, and thus the inherited column is ticked.

Reference is now made to FIG. 27, which is an illustration of displaying a relation within an explorer-type viewing tool, in accordance with a preferred embodiment of the present invention. FIG. 27 includes a screen 2710 having a left panel 2720 with an explorer-type user interface, and a right panel 2730 for displaying a text box 2740 with a list of classes in the domain of the relation. Screen 2710 indicates that a relation Written_By includes classes Article and Author in its domain. The Written_By relation and the classes Article and Author belong to the namespace Newspaper.Journalism.

Reference is now made to FIG. 28, which is an illustration of a tool for viewing ontologies, in accordance with a preferred embodiment of the present invention. FIG. 28 shows a screen 2810 including a text box 2820 for displaying a list of ontology models, and a text box 2830 for displaying a list of versions within a selected model displayed in text box 2820. FIG. 28 also

includes a button 2840 for launching a search tool, as described below with reference to FIG. 29.

Reference is now made to FIG. 29, which is an illustration of a search tool for finding a desired ontology, in accordance with a preferred embodiment of the present invention. FIG. 29 shows a screen 2910 including a text box 2920 for displaying an ontology search query, a text box 2930 for displaying the results of the query as a list of ontology models, and a text box 2940 for displaying a list of versions within a selected model displayed in text box 2930. Screen 2910 shows the results of a query for ontology models whose names contain the word IT.

Reference is now made to FIG. 30A and 30B, which are illustrations of a tool for displaying and editing enumerated values of a class, in accordance with a preferred embodiment of the present invention. FIG. 30A shows a screen 3010 including a text box 3020 for selecting enumerated values for a class. FIG. 30B shows a screen 3030 including a text box 3040 for editing an enumerated value.

Examples

Examples of enterprise applications of distributed ontologies abound. The following is a simple example corresponding to FIG. 1.

A routine invoice is received from TRR, Inc. totalling $482.83. The 14 items listed in the invoice include 3 laptop computers, 1 CD-ROM and 10 3-packs of 8mm cassettes. Information about TRR as a customer and their contact information, the payment terms for the $482.83, and detailed information about each of the three individual products are also available directly from the invoice. These categories are located within a larger abstract class called Companies, by viewing a relation Customer that describes the instance TRR within the class Companies. TRR customer information allows the user to view additional information about TRR or about other companies, including the ability to access the class of Dun & Bradstreet Credit Ratings. The D&B Credit Rating defines a relation for all enterprises between the classes Companies and Credit Ratings.

TRR employee information can be viewed by viewing a relation called Employees between the classes Companies and People. Once this is accomplished, the user views the relations of TRR employees to the superclass People. Possible attributes for the superclass People include date of birth, first name, middle initial, last name and identification number.

The payment terms for the current invoice relate to a class Future Revenues, and provide details about the overall future estimated performance of a company and how this is compiled.

Each individual line item listed in the TRR invoice is an instance of a class Products, which includes attributes SKU (Stock Keeping Unit), Manufacturer, Dimensions, Weight, and Standards. The classes Components and Measurements are separate classes related to the Products class through the relations Bills of Materials and Technical Specifications, respectively.

A syntax for the above example is as follows:

Classes

- Laptop Computers
- CD-ROMS
- 8mm cassettes × 8mm cassettes × 8mm cassettes
- Corporations
- Addresses
- Real Numbers
- Character Strings
- Latin Letters
- D&B Credit Ratings
- People
- Dates
- Integers
- Products
- Invoices
- Legal Entities
- Future Revenues

Relations

- Customer_of ⊆ Corporations × Legal Entities
- Employee_of ⊆ Corporations × Persons

Functions

- Invoice_Items: Invoices → **List**[Products]
- Price_in_US_dollars: Products → Real Numbers
- Total_$_invoice_bill: Invoices → Real Numbers
- Credit_Rating: Legal Entities × Date → D&B Credit Ratings
- Date_of_birth: People → Dates
- Given_name: People → Character Strings

- Surname: People → Character Strings
- Middle_initial: People → Latin Letters
- ID_number: People → Integers
- Payment_terms: Invoices → Future Revenues
- SKU: Products → Integers
- Manufacturer: Products → Legal Entities
- Dimensions_in_centimeters^3: Products → Real Numbers^3
- Weight_in_kilograms: Products → Real Numbers
- bill_of_materials: Products → Products
- Cardinality_of_item_within_invoice: Invoices × Products → Integers
- Address_of: Legal Entities → Addresses
- Telephone_number: Legal Entities → Integers

Inheritance

- Corporations ⊆ Legal Entities
- People ⊆ Legal Entities
- Laptop Computers ⊆ Products
- 8mm Cassettes ⊆ Products
- Addresses ⊆ Character Strings
- Integers ⊆ Real Numbers

Reference is now made to Appendix A, which is a simplified listing of three example ontologies, in accordance with a preferred embodiment of the present invention. Appendix A includes examples of ontologies for (i) motor cars, (ii) airline travel, and (iii) purchase orders. Each example is described using an XML document according to an XML syntax, which has been developed for defining ontologies. While this particular syntax provides a central definition, the same syntax can be used in a distributed environment with remote classes referred to by URLs of defining documents together with local names of the classes within such document.

The example motor car ontology includes the following classes and relations:

Classes

1. Cars
2. CarManufacturers ⊆ LegalEntities
3. Models
4. Persons ⊆ LegalEntities

5. LegalEntities
6. FuelTypes
7. TireTypes
8. Contracts
9. TransmissionTypes
10. BrakeSystems
11. EngineTypes
12. Distances
13. Speeds

Relations

1. owner	$\subseteq$ Cars × LegalEntities
2. insuranceCarrie	$\subseteq$ Cars × LegalEntities
3. insurancePolicy	$\subseteq$ Cars × LegalEntities × Contracts

Functions

1. make:	Cars → CarManufacturers
2. fuelIntake:	Cars → FuelTypes
3. color:	Cars → Colors
4. tires:	Cars → TireTypes
5. tireManufacturers:	TireTypes → LegalEntities
6. transmission:	Cars → TranmissionTypes
7. mileage:	Cars → Distances
8. maximumSpeed:	Cars → Speeds

The example airline travel ontology includes the following classes and relations:

Classes

1. LegalEntities
2. Airlines $\subseteq$ LegalEntities
3. Airports
4. Persons $\subseteq$ LegalEntities
5. Trips
6. Locations
7. Flights
8. Airplane Types

Relations

1.	travelers	$\subseteq$ Trips $\times$ Persons
2.	destinations	$\subseteq$ Trips $\times$ Locations

Functions

1.	carrier:	Flights $\rightarrow$ Airlines
2.	takeoff:	Flights $\rightarrow$ Airports
3.	landing:	Flights $\rightarrow$ Airports
4.	travelAgent:	Trips $\rightarrow$ LegalEntities
5.	airplane:	Flights $\rightarrow$ AirplaneTypes
6.	connectingFlight:	Trips $\times$ Persons $\times$ Flights $\rightarrow$ Flights

The example airline travel ontology includes the following classes and relations:

Classes

1. LegalEntities
2. PurchaseOrders
3. Corporations $\subseteq$ LegalEntities
4. StockItems
5. Persons $\subseteq$ LegalEntities
6. Addresses

Relations

1.	itemsPurchased	$\subseteq$ PurchaseOrders $\times$ StockItems
2.	providers	$\subseteq$ PurchaseOrders $\times$ LegalEntities

Functions

1.	shipTo:	PurchaseOrders $\rightarrow$ Addresses
2.	customers:	PurchaseOrders $\rightarrow$ LegalEntities
3.	billTo:	PurchaseOrders $\rightarrow$ Addresses

In a preferred embodiment of the present invention, the basic elements of an ontology are expressed within an "Ontology XML" document Reference is now made to Appendix B, which is a simplified listing of an XML Schema for an Ontology XML document, in accordance with a preferred embodiment of the present invention.

As can be seen in Appendix B, a general ontology model preferably includes classes, complex classes, packages, relations, function and

inheritances. An ontology has attributes; i.e., meta-data of the model, for a required version and for optional language and author.

As can be seen in Appendix B, provision is made for both classes and complex classes. Generally, complex classes are built up from simpler classes using tags for symbols such as intersection, Cartesian product, set, list and bag. The "intersection" tag is followed by a list of classes or complex classes. The "Cartesian product" tag is also followed by a list of classes or complex classes. The "set" symbol is used for describing a class comprising subsets of a class, and is followed by a single class or complex class. The "list" symbol is used for describing a class comprising ordered subsets of a class; namely, finite sequences, and is followed by a single class or complex class. The "bag" symbol is used for describing unordered finite sequences of a class, namely, subsets that can contain repeated elements, and is followed by a single class or complex class. Thus **set**[C] describes the class of sets of instances of a class C, **list**[C] describes the class of lists of instances of class C, and **bag**[C] describes the class of bags of instances of class C.

In terms of formal mathematics, for a set S, **set**[S] is **P**(S), the power set of S; **bag**[S] is N^S, where N is the set of non-negative integers; and **list**[S] is $\bigcup_{n=1}^{\infty} S^n$. There are natural mappings

$$\textbf{list}[S] \xrightarrow{\phi} \textbf{bag}[S] \xrightarrow{\psi} \textbf{set}[S]. \qquad (1)$$

Specifically, for a sequence $(s_1, s_2, \ldots, s_n) \in \textbf{list}[S]$, $\phi(s_1, s_2, \ldots, s_n)$ is the element $f \in \textbf{bag}[S]$ that is the "frequency histogram" defined by $f(s) = \#\{1 \le i \le n: s_i = s\}$; and for $f \in \textbf{bag}[S]$, $\psi(f) \in \textbf{set}[S]$ is the subset of S given by the support of f, namely, $\text{supp}(f) = \{s \in S: f(s) > 0\}$. It is noted that the composite mapping $\phi\psi$ maps a the sequence $(s_1, s_2, \ldots, s_n)$ into the set of its elements $\{s_1, s_2, \ldots, s_n\}$. For finite sets S, **set**[S] is also finite, and **bag**[S] and **list**[S] are countably infinite.

Attributes of a class and attributes of a complex class preferably include (i) a required label for the class to be used for all references within the XML document; (ii) an optional label for instances of the class, which may be distinct from the label of the class; (iii) an optional URL to an icon; and (iv) an optional name of a package. A base class, Being, from which all classes inherit, is included.

Attributes of a relation include (i) a domain, which is a complex class; (ii) a required label for the relation to be used for all references within the XML document; (iii) an optional URL to an icon; and (iv) an optional name of a package. Attribute of functions include (i) a domain and a co-domain, each of which is a complex class; (ii) a required label for the relation to be used for all

references within the XML document; (iii) an optional URL to an icon; and (iv) an optional name of a package.

Attributes of an inheritance include a pairType, which includes a class reference for a subclass, and a class reference for a superclass.

As can be seen in Appendix B, the main body of an Ontology XML document is preferably composed of three to five main sections. A declaration of the ontology's classes, a declaration of relations, and a declaration of functions are preferably required. Sections declaring certain complex classes and explicitly spelling out inheritance relations between classes may optionally be included as well.

Although the Schema in Appendix B anticipates inclusion of all definitions in one document, the same syntax is used to publish a fragment of an ontology on a distributed ontology web. To effect this, a full URL plus class name are preferably used instead of a local IDREF, to refer to a class.

It may be appreciated by those skilled in the art that an ontology stored in an XML document according to this Schema can easily be parsed using a program written using a parser, such as a parser with a DOM or SAX interface, and converted into a Java or C++ object oriented representation of the ontology model.

Header

There are certain elements that preferably appear in the header of an ontology document. At the top, as in all XML documents, the version of XML being used is declared, as follows:

<?xml version="1.0" encoding="UTF-8"?>

This declaration is preferably followed by an opening tag with the word "Ontology" followed by conventional XML namespace declarations and a pointer to the official copy of the ontology XML Schema:

<Ontology
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="\\Calvin\ABC-Files\Departments\R-D\Releases\Release 0.1 Documents\OntSchema1.xsd"

Two preferably required fields follow: the version of the schema being used, and a description of the overall subject field that this particular ontology is modeling.

ABCSchemaVersion="0.1"

ontologyOf="Invoices"

The header preferably closes with two optional fields: one naming the author of the document, and one declaring the main language being used in the document. A closing tag indicates the end of the header.

author="Fred Bloggs"
lang="en-UK">

Optionally, a general comment regarding the ontology is added at this point:

<comment> This is my first ontology </comment>

More than one comment may appear, and comments may appear in different languages and identified as such:

<comment lang="en"> That's it </comment>
<comment lang="fr"> C'est ça </comment>

All comment, description and example elements, wherever they appear in an ontology XML document, may appear more than once and in various languages.

Classes Declaration

The first block of the main body of an ontology XML document is a listing of declared classes. The simplest way of listing them is in the following style:

<classesDeclaration>
<class classLabel="CD-ROMS"/>
<class classLabel="Persons"> </class>
<class classLabel="Legal_Entities"/>
<class classLabel="Corporations"/>
<class classLabel="Car_Manufacturers"/>
<class classLabel="Products"/>
<class classLabel="Real_Numbers"/>
<class classLabel="Character_Strings"/>
<class classLabel="Invoices"/>
<class classLabel="Dates"/>

<class classLabel="Credit_Ratings"/>
</classesDeclaration>

The opening tag is of the form <classesDeclaration>. It is followed by a list of <class> </class> tags, with a required attribute classLabel, containing a unique ID for each class. The </classesDeclaration> closes this block. At minimum, every classes declaration block preferably contains these elements.

The ClassesDeclaration block can be commented, if so desired, by inserting a comment tag as follows:

<comment> This is a list of classes for explanatory purposes. </comment>

Each individual class can similarly be commented:

<class classLabel="Corporations" >
<comment lang="en"> This is here to show how comments may be added. </comment>
</class>

Separate elements for general descriptions and examples are also provided. Subclass and superclass relations can be included within appropriate class tags. In addition to the official "classLabel", numerous "userLabels" may also be added as attributes to a class, to indicate synonymous names that can be used interchangeably by different users to refer to the class. Finally, apart from the general names for a class, it may be desirable to use a different name for referring to particular instances. This is done by adding the attribute "instanceLabel":

<class classLabel="Corporations" userLabel="Companies"
instanceLabel="Company">
<comment lang="en"> This is here to show how comments may be added. </comment>
<description> Entities doing business. </description>
<examples> General Motors, IBM, Panasonic </examples>
<subClassOf label="Legal_Entities" />
<superClassOf label="Car_Manufacturers" />
</class>

Complex Classes Declaration

Preferably, the complex classes declaration block is optional. It may be convenient for use when one makes frequent use of a particular complex class in domains and co-domains, because it enables reference to that complex class by use of a single label instead using an in-line description of the complex-class' structure repeatedly.

The overall syntax for a complex classes declaration block is similar to that of the classes declaration block:

```
<complexClassesDeclaration>
    <complexClass CClabel="setsOfPersons" userLabel="groupsOfHumans">
        <set>
            <classRef label="Persons"/>
        </set>
    </complexClass>

    <complexClass CClabel="lists_of_sets_of_corporations">
        <list>
            <set>
                <classRef label="Corporations"></classRef>
            </set>
        </list>
    </complexClass>
</complexClassesDeclaration>
```

Each complex class preferably has a CCLabel, similar to the classLabel assigned to classes, in order to provide unique identification. User labels may be added here as well.

Every complex class preferably opens with one of the tags <set>, <bag>, <list>, <intersection> or <Cartesian>. These tags may be nested as deeply as desired and in any order. The innermost layer of the nesting preferably contains a tag <classRef> with an attribute Label, whose value is preferably one of the class labels or use labels used to identify a class within the ClassesDeclaration block.

Descriptions and examples may be added here, just as they may in the ClassesDeclaration block.

Relations Declaration

Preferably, the relations declaration block is required, although it may be empty if one does not use relations in the ontology.

```
<relationsDeclaration>
    <relation relationLabel="customerOf">
        <domain>
            <classRef label="Corporations"/>
            <classRef label="Legal_Entities"/>
        </domain>
    </relation>

    <relation relationLabel="employee_of">
        <domain>
            <classRef label="Corporations"/>
            <classRef label="Persons"/>
        </domain>
    </relation>
</relationsDeclaration>
```

Every relation preferably has a "relationLabel" attribute – with an optional userLabel attribute. The <relation> tag is preferably followed by one and only one <domain> tag child element. The children elements of the <domain> tag are preferably a list of <classRef> tags, each with an obligatory "label" attribute whose value preferably is a labeled class or a complex-class. As above, comments, descriptions and examples, in multiple languages, may be freely added.

Functions Declaration

Preferably, the functions declaration block is required, although it may be empty if one does not use relations in the ontology.

```
<functionsDeclaration>
    <function functionLabel="princeInUSDollars">
        <domain>
            <classRef label="Products"/>
        </domain>
        <range>
            <classRef label="Real_Numbers"/>
        </range>
```

```
    </function>

    <function functionLabel="Surname">
      <domain>
        <classRef label="Persons"/>
      </domain>
      <range>
        <classRef label="Character_Strings"/>
      </range>
    </function>
</functionsDeclaration>
```

Every function preferably has a "functionLabel" attribute, with an optional userLabel attribute. The <function> tag is preferably followed by a <domain> tag child element and a <range> tag child element. The children elements of the <domain> element are preferably a list of <classRef> tags, each with an obligatory Label attribute whose value preferably is only a labeled class or complex-class. The same restriction preferably holds for the children of the <range> element. As always, comments, descriptions and examples, in multiple languages, may be freely added.

It may occur that the domain or co-domain is a complex class; for example, when a function associates each invoice with a list of products appearing on the invoice. In such a case, one of two options may be used.

The first option is to describe the complex structure of the domain or range in-line, as follows:

```
<function functionLabel="invoiceItems">
  <domain>
    <classRef label="Invoices"/>
  </domain>
  <range>
    <list>
      <classRef label="Products"/>
    </list>
  </range>
</function>
```

The second option is to declare in the ComplexClassesDeclaration block the existence of a complex class for lists of

Products, give it a label such as Product_List, and then use that label as the value of the classRef label attribute:

```
<function functionLabel="invoiceItems">
    <domain>
        <classRef label="Invoices"/>
    </domain>
    <range>
        <classRef label="Product_List"/>
    </range>
</function>
```

Inheritance Declaration

Preferably, the inheritance declaration block is optional. It enables the concentrated expression of the subclassing tree. For example:

```
<inheritanceDeclaration>
    <inheritancePair>
        <subclass label="Corporations"/>
        <superclass label="Legal_Entities"/>
    </inheritancePair>
</inheritanceDeclaration>
```

Preferably, every inheritance declaration is a list of <inheritancePair> elements. Preferably, every <inheritancePair> element is followed by two children tag elements: <subclass> and <superclass>, in that order. The <subclass> and <superclass> elements are empty elements, in the sense that they have neither child elements nor text within them. They each carry a Label attribute, with a value from among the declared class labels.

Closing the Document

The closing tag preferably is:

```
</Ontology>
```

Views

Formally, a view, v, of a class, c, is defined recursively to be a set of pairs $\{(f_s, v_s): s \in S\}$, indexed by a set S, and a usability rule **U**, where each f_s is a function with domain c, each v_s is a view of the co-domain of f_s, and **U** is a

subset of **list**[S]. Specifically, **U** includes those sequences (s1, s2, ..., sn) for which the corresponding sequence of functions and views $((f_{s1}, v_{s1}), (f_{s2}, v_{s2}), \ldots, (f_{sn}, v_{sn}))$ make the view v usable, i.e., acceptable.

If $\mathbf{U} = \psi^{-1}(\mathbf{U'})$, for some subset **U'** of **bag**[S], where ψ is the mapping from **list**[S] to **bag**[S] defined above with reference to Eq. (1), then the function–view pairs (f_s, v_s) are unordered; i.e., they can be prescribed in any order and **U** can be identified with **U'**. If $\mathbf{U'} = \phi^{-1}(\mathbf{U''})$ for some subset **U''** of **set**[S], where ϕ is the mapping from **bag**[S] to **set**[S] defined above with reference to Eq. (1), then repetitions of function-view pairs (f_s, v_s) can be ignored and **U** can be identified with **U''**.

To facilitate simple descriptions of a usability rule, syntax is introduced, similar to the familiar xsd:all, xsd:sequence, xsd:choice and minOccurs / maxOccurs syntax used in XML to describe lists of nametags. Specifically, the term "required" is used with an individual pair (f, s) to indicate that it is required in the description of the view, and the term "optional" is used to indicate that (f, s) is not required. The term "strict" is used to indicate that each pair (f_s, v_s) is required once, in any order. In this case, $\mathbf{U} = \{S\}$. The term "liberal" is used to indicate that each pair (f_s, v_s) is optional. In this case **U** = **set**[S]. The term "choice" is used to indicate a choice of one item from among a plurality of items. The term "ordered" is used to indicate that the pairs (f_s, v_s) are ordered.

Views can also extended from classes to container classes. Specifically, if v is a view for a class C, then a view for **set**[C], **bag**[C] or **list**[C] can be defined as a plurality of views of C.

Reference is now made to FIG. 31A, which is a diagram of the relationship between a view of a class and a description of an instance and, correspondingly, an XML Schema and an XML document, in accordance with a preferred embodiment of the present invention. Converting a view to a complex type within an XML Schema is preferably done by converting each of the pairs (f_s, v_s) to an <xsd:element> and by using the XML constraints xsd:all, xsd:sequence, xsd:choice and minOccurs / maxOccurs to arrange the elements according to the usability rule **R**.

Thus if (f_s, v_s) is a required pair then the corresponding XML element is

<xsd:element name="f_s" type="v_s" />

and if (f_s, v_s) is an optional pair then the corresponding XML element is

<xsd:element name="f_s" type="v_s" minOccures="0" />.

A description of an instance $x \in C$ based on a view, v, of C as described above, is defined recursively to be a set of pairs $\{(f_t(x), w_t): s \in T\}$,

where each f is a function with domain C, each w_t is a description of $f(x)$, and $T \in \mathbf{R}$. Descriptions of instances can be converted to XML documents in an analogous way that views of classes are converted to XML Schema.

Reference is now made to FIG. 31B, which is a UML diagram corresponding to FIG. 31A, in accordance with a preferred embodiment of the present invention. FIG. 31B indicates that each view corresponds to one class, but one class can have multiple views. Similarly, each description corresponds to one view, but one view can have multiple descriptions.

Examples of conversion of views to XML Schema are provided in Appendices C1 – C6. Each of these examples also contains a valid XML document conforming to the XML Schema, corresponding to a description of an instance.

Examples C1 – C6 concern views of a class Books within an ontology. Each example includes four functions -- namely, Author, Title, ISBN and Publisher; and illustrates a different usability rule. The usability in Example C1 is strict, in that each of the four function views is required. The view in Example C1 also indicates the views used for each of Author, Title, ISBN and Publisher, and, in turn, the views used for those views.

The usability in Example C2 is liberal, in that each of the four function views is optional. The usability in Example C3 is mixed, in that views for Title and ISDB are required, but views for Author and Publisher are optional. The usability in Example C4 is ordered. The usability in Example C5 specifies the constraint syntax "Choice" to indicate that two choices are required, and a view of Publisher is optional. The first choice is either a view of Author or else a view of Title, and the second choice is either a view of ISBN or else a view of Library_Cat_Num.

The usability in Example C6 includes a container view whereby the view for Author uses a view for the class **set**(Persons), since a book may have more than one author. The view for **set**(Persons) is indicated by **set**(Persons, v_1), where v_1 is a view for Persons. The view **set**(Persons, v_1) indicates a set of individual views of persons, using view v_1 for each individual view.

In reading the above description, persons skilled in the art will realize that there are many apparent variations that can be applied to the methods and systems described.

It will be appreciated by persons skilled in the art that the present invention is not limited by what has been particularly shown and described hereinabove. Rather the present invention includes combinations and sub-combinations of the various features described hereinabove as well as modifications and extensions thereof which would occur to a person skilled in the

art and which do not fall within the prior art.